Міністерство освіти та науки України
Національний технічний університет України
«Київський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ

**до виконання лабораторних робіт
з дисципліни «Системи технічного зору»**

для студентів спеціальностей
7.05090201 та 8.05090201 «Радіоелектронні апарати та засоби»

Рекомендовано
Вченою радою факультету
електроніки

Протокол № 9/2015
від 28 вересня 2015 р.

Рекомендовано
кафедрою конструювання електронно-
обчислювальної апаратури

Протокол № 12 від 09 вересня 2015 р.

Завідувач кафедри КЕОА

_____ О.М.Лисенко

Київ – 2015

Методичні вказівки до виконання лабораторних робіт з дисципліни «Системи технічного зору» для студентів спеціальностей 7.05090201, 8.05090201 «Радіоелектронні апарати та засоби» / Уклад.: А.Ю. Варфоломєєв, В.Г. Дзюба – К.: НТУУ «КПІ», 2015. – 83 с.

ЕЛЕКТРОННЕ НАВЧАЛЬНЕ ВИДАННЯ

Методичні вказівки

до виконання лабораторних робіт
з дисципліни «Системи технічного зору»
для студентів спеціальностей
7.0590201 та 8.05090201 «Радіоелектронні апарати та засоби»

Укладачі: Варфоломєєв Антон Юрійович, к.т.н., ст. викладач
Дзюба Віталій Георгійович, к.т.н.

Відповідальний редактор: Лисенко Олександр Миколайович
докт. техн. наук, проф.

За редакцією укладачів

ЗМІСТ

ВСТУП

Системи технічного зору на сьогодні набули великого розповсюдження та знаходять все ширше застосування у різноманітних галузях людської діяльності. Технічний або комп'ютерній зір як галузь науки розвивається вже протягом кількох десятиліть і поступово накопичені знання перетікають у дисципліни, що викладаються в університетах та інститутах. Це обумовлено тим, що попит на спеціалістів, які володіють навичками реалізації складних алгоритмів обробки статичних зображень та відеопослідовностей постійно зростає.

Важлива особливість сучасного підходу до створення систем технічного зору полягає у швидкому переході від ідеї до математичного алгоритму, перевірка якого не потребує значних затрат. Знадобиться лише ПК та найпростіша веб-камера.

Процес розробки систем технічного зору є досить інтерактивним та наочним. Водночас, він не позбавлений ряду складностей, які головним чином обумовлені створенням та реалізацією алгоритмів, які власне і виконуватимуть обробку зображень та/або відеопослідовностей. Допомогти подолати зазначені складності, спрощуючи процес розробки, можуть інтегровані середовища та спеціалізовані бібліотеки, яких нині створено чимало.

Серед них слід відзначити систему комп'ютерної математики MATLAB, до складу якої входять модулі (toolboxes), що дозволяють не тільки отримувати зображення та виконувати їх первинну обробку, але й проводити складний аналіз відеоданих, який може включати: морфологічні операції, сегментацію, статистичний аналіз, розпізнавання образів за допомогою нейронних мереж, тощо. З огляду на особливу зручність системи MATLAB у даній роботі всі методи обробки зображень проілюстровані на прикладі процедур та команд саме цієї системи. Ще однією перевагою MATLAB є можливість його складової – середовища SIMULINK генерувати програмний код для ряду вбудованих систем,

створених на основі цифрових сигнальних процесорів – DSP (мовою C/C++) та програмованих логічних матрицях (на мовах VHDL/Verilog). Окрім MATLAB необхідно також згадати і бібліотеку OpenCV, частково розроблену компанією Intel. Дана бібліотека написана мовою C/C++, що забезпечує їй високу швидкодію та низьку ресурсоємність у порівнянні з системою MATLAB. До того ж OpenCV є вільно розповсюджуваною і має відкритий код.

Слід зазначити, що останнім часом спостерігається цікава тенденція – велика кількість програм, орієнтованих на обробку зображень і зокрема розпізнавання образів переносяться на вбудовані (embedded) та мобільні системи з метою розробки новітніх інтерфейсів між людиною та машиною. З огляду на цей факт, вдосконалення та подальший розвиток систем технічного зору набувають особливої актуальності.

ЛАБОРАТОРНА РОБОТА №1

Просторові методи обробки зображень

Тема роботи: просторові методи обробки зображень.

Мета роботи: реалізувати основні просторові методи обробки зображень в середовищі MATLAB або Python.

Теоретичні відомості

Просторові методи обробки зображень – це процедури, що оперують безпосередньо значеннями пікселів зображення. Процес просторової обробки описується наступним рівнянням:

$$g(x, y) = T[f(x, y)],$$

де $f(x, y)$ – вхідне зображення, $g(x, y)$ – оброблене зображення, а T – оператор над f, визначений в деякому околі точки (x, y). Якщо окіл, в якому визначено оператор T має розмір 1×1, то такий вид просторової обробки називають *градаційним*. Якщо ж окіл має прямокутну або квадратну форму, то таку просторову обробку називають *віконною*. Віконна обробка може бути *лінійною* – виконуватись за допомогою операції двомірної згортки або *нелінійною* – передбачати застосування деякого нелінійного оператора до околу.

Найбільш поширені види градаційного перетворення

- Перетворення у негатив.
- Логарифмічне перетворення.
- Ступеневе перетворення.
- Розтягнення контрасту.

  Розтягнення контрасту у MATLAB також можна виконати за допомогою функції **`imadjust`**.
- Еквалізація гістограм.

Еквалізація гістограм в системі MATLAB виконується за допомогою функції **`histeq`**. Пошук та відображення на екрані гістограм зображень можна виконати за допомогою функції **`imhist`**. У Python еквалізація гістограм може бути здійснена за допомогою функції **`equalize_hist`** пакету **`skimage.exposure`**. Для виведення гістограми зображення **`im`** можна скористатися функцією **`hist(im.ravel(), bins=256, range=(0,255))`** пакету **`matplotlib.pyplot`**.

Основні види віконних перетворень

Віконні перетворення виконуються за допомогою операції двомірної згортки:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t),$$

де $a = (m-1)/2$ та $b = (n-1)/2$, w – масив (маска), що містить ядро фільтру. В залежності від того, якою буде маска, можна отримати:

- Лінійне згладжування зображень.

  Найчастіше використовують однорідні усереднюючі фільтри та фільтри зі зваженим середнім (див. лекцію).
- Лінійне підвищення різкості зображень.

  Досягається за рахунок віднімання від вихідного зображення результату його фільтрації оператором Лапласа (другої похідної зображення).
- Виділення для перепадів яскравості.

  Досягається за рахунок застосування до зображення операторів Собела, Превіта, Робертса, Кірша (див. лекцію).
- Нелінійна фільтрація.

  Найбільш розповсюдженою фільтрацією цього виду є медіанна фільтрація.

Для лінійних віконних методів перетворення, масив *w* можна задати як вручну, так і отримати автоматично, скориставшись функцією **`fspecial`**.

Для виконання віконної фільтрації зображень в MATLAB передбачена функція **`imfilter`**. У Python віконну фільтрацію можна здійснити функцією двовимірної згортки **`convolve2d`** , що реалізована в пакеті **`scipy.signal`**.

Нелінійна медіанна фільтрація в системі MATLAB виконується за допомогою функції **`medfilt2`**. У Python для цього можна скористатись функцією **`medfilt`** пакету **`scipy.signal`**.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові m-файли з функціями надаються окремо).
2. Провести експериментальні дослідження впливу параметрів функцій обробки на якість результуючих зображень.
3. Представити процедури обробки зображень у вигляді m-файла.

Завдання

Здійснити над заданими зображеннями:

1. Перетворення в негатив (файл – pic1.jpg).
2. Логарифмічне перетворення (файл – pic2.jpg).
3. Степеневе перетворення (файл – pic3.jpg).
4. Розтягнення контрасту (файл – pic4.jpg).
5. Еквалізацію гістограм (файл – pic5.jpg).
6. Згладжування усереднюючим фільтром (файл – pic6.jpg).
7. Підвищення різкості з використанням маски Лапласа (файл – pic7.jpg).
8. Градієнтну обробку (файл – pic8.jpg).
9. Медіанну фільтрацію (файл – pic9.jpg).

З’ясувати як впливають параметри, що застосовуються в кожному виді фільтрації на її якість.

Запитання для самоконтролю

1. Наведіть класифікацію методів просторової обробки зображень.
2. Що таке градаційне перетворення зображень.
3. Які особливості логарифмічного перетворення, навіщо в ньому передбачено додавання до зображення 1.
4. В чому полягають особливості перетворення основаного на розтягненні контрасту.
5. Що таке віконне перетворення зображень, на чому ґрунтується.
6. Під час віконного перетворення, результуюче зображення виходить дещо меншим ніж вихідне. Які засоби застосовуються для того, щоб вказаний ефект подолати (див. допомогу до функції **`imfilter`**).
7. В чому полягає суть фільтра Лапласа. Як цю маску можна отримати.
8. Що таке медіана, як виконується обробка зображень з її допомогою.

ЛАБОРАТОРНА РОБОТА №2

Частотні методи обробки зображень

Тема роботи: частотні методи обробки зображень в системі MATLAB.

Мета роботи: реалізувати основні частотні методи обробки зображень в середовищі MATLAB або Python.

Теоретичні відомості

Для отримання образу зображення в частотній області використовується двомірне дискретне перетворення Фур'є:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi(ux/M+vy/N)}$$

де f – вихідне зображення розміром $M \times N$; F – його образ в частотній області; x та y – просторові змінні; u та v – частотні змінні.

Зворотне перетворення Фур'є виконується за формулою:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{i2\pi(ux/M+vy/N)}.$$

Спектр $|F(u,v)|$, *фаза* $\varphi(u,v)$ і *енергетичний спектр* $P(u,v)$ визначаються відповідно:

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)},$$

$$\varphi(u,v) = arctg\left[\frac{I(u,v)}{R(u,v)}\right],$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v).$$

Фільтрація зображень в частотній області

Фільтрація в частотній області виконується на основі теореми про згортку:

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v)H(u,v),$$

де f та F – зображення та його образ у частотній області, отриманий за допомогою перетворення Фур'є; h та H – фільтр та його образ у частотній області відповідно. Подвійна стрілка вказує на те, що вираз зліва (просторова згортка) може бути отриманий застосуванням *зворотного* перетворення Фур'є до виразу справа (добуток $F(u, v)\cdot H(u, v)$ в частотній області) і, навпаки, вираз справа може бути отриманий застосуванням *прямого* перетворення Фур'є до виразу зліва.

Узагальнений алгоритм фільтрації зображень в частотній області

1. Отримати параметри розширення за допомогою **`paddedsize`**:

```
PQ = paddedsize(size(f));
```

2. Побудувати перетворення Фур'є з розширенням:

```
F = fft2(f, PQ(1), PQ(2));
```

3. Згенерувати функцію фільтра **`H`** розміром **`PQ(1) * PQ(2)`**:

```
H = lp_filter(...);    % НЧ фільтр
H = hp_filter(...);    % ВЧ фільтр
```

4. Помножити перетворення Фур'є на передатну функцію фільтра:

```
G = H .* F;
```

5. Знайти дійсну частину зворотного перетворення Фур'є від **`G`**:

```
g = real(ifft2(G));
```

6. Вирізати верхній лівий прямокутник вихідних розмірів:

```
g = g(1:size(f, 1), 1:size(f, 2));
```

Для зручності, процедура фільтрації за заданим відгуком фільтра (кроки 2, 4, 5 та 6) реалізована у функції **`dftfilt`** (див. [2]). Відгуки фільтрів у частотній області можна згенерувати автоматично функціями **`lp_filter`** та **`hp_filter`**. При роботі з Python всі відповідні функції реалізовано у пакеті dftfilt.py.

Відгуки основних фільтрів в частотній області

Табл. 2.1. Відгуки фільтрів в частотній області.

Тип	НЧ	ВЧ
Ідеальний	$H(u,v)=\begin{cases}1, \text{ при } D(u,v)\le D_0\\ 0, \text{ при } D(u,v)>D_0\end{cases}$	$H(u,v)=\begin{cases}0, \text{ при } D(u,v)\le D_0\\ 1, \text{ при } D(u,v)>D_0\end{cases}$
Баттерворта	$H(u,v)=\dfrac{1}{1+\left[\frac{D(u,v)}{D_0}\right]^{2n}}$	$H(u,v)=\dfrac{1}{1+\left[\frac{D_0}{D(u,v)}\right]^{2n}}$
Гауса	$H(u,v)=\exp\left[\frac{-D^2(u,v)}{2D_0{}^2}\right]$	$H(u,v)=1-\exp\left[\frac{-D^2(u,v)}{2D_0{}^2}\right]$
Лапаласа	–	$H(u,v)=-(u^2+v^2)$

де $D(u, v) = [(u - M/2)^2 + (v - N/2)^2]^{1/2}$;

D_0 – частота (радіус) зрізу фільтрів;

n – порядок фільтру Баттерворта.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові m-файли з функціями надаються окремо).
2. Провести експериментальні дослідження впливу параметрів фільтрації на якість результуючих зображень.
3. Представити процедури обробки зображень у вигляді m-файла.

Завдання

Виконати над заданими зображеннями:

1. НЧ фільтрацію ідеальним фільтром (файл – pic1.jpg);
2. НЧ фільтрацію фільтром Баттерворта (файл – pic1.jpg);
3. НЧ фільтрацію фільтром Гауса (файл – pic1.jpg);
4. ВЧ фільтрацію ідеальним фільтром (файл – pic1.jpg);

5. ВЧ фільтрацію фільтром Баттерворта (файл – pic1.jpg);
6. ВЧ фільтрацію фільтром Гауса (файл – pic1.jpg);
7. ВЧ фільтрацію лапласіаном (файл – pic1.jpg);
8. Виконати порівняння фільтрації в просторовій та частотній областях (файл – pic2.jpg);

Запитання для самоконтролю

1. На чому основана фільтрація зображень в частотній області.
2. Як на перетвореному зображенні-образі розташовані частоти.
3. В чому полягає суть функції **`fftshift2`**, навіщо потрібна ця функція, і коли вона має застосовуватись.
4. Наведіть повну процедуру фільтрації в частотній області.
5. Навіщо потрібне розширення зображень **`paddedsize`**.
6. Наведіть основні види НЧ-фільтрів.
7. Наведіть основні види ВЧ-фільтрів.
8. Що таке ефект «дзвону». Коли він виникає.
9. Як перейти від НЧ-фільту до відповідного ВЧ-фільтру і навпаки.
10. Який вид фільтрації – частотна чи просторова є більш ефективною і коли.

ЛАБОРАТОРНА РОБОТА №3

Відновлення зображень

Тема роботи: відновлення зображен.

Мета роботи: реалізувати основні методи відновлення зображень в середовищі MATLAB або Python.

Теоретичні відомості

Процес спотворення зображень можна описати наступним рівнянням:

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y),$$

де h – спотворюючий оператор; η – адитивний шум, «*» – позначає згортку. Еквівалентне представлення в частотній області має вигляд:

$$G(u, v) = H(u, v) \cdot F(u, v) + N(u, \text{v}),$$

де H, N – відповідні частотні образи.

Періодичний шум зручно моделювати двомірною синусоїдою, що має вигляд:

$$r(x, y) = A\sin\left[\frac{2\pi u_0 (x + B_x)}{M} + \frac{2\pi v_0 (y + B_y)}{N}\right],$$

де A – амплітуда, u_0 та v_0 – частоти по осям x та y відповідно, B_x та B_y – зсуви фаз відносно початку відліку по осям x та y відповідно.

Найпоширеніші моделі шуму

Серед найбільш поширених моделей шуму можна виділити наступні:

- Рівномірний шум.
- Шум із розподілом Гауса.
- Логарифмічний нормальний шум.
- Експоненційний шум.
- Шум із розподілом Релея.
- Шум із розподілом Ерланга.

В даній роботі для моделювання наведених вище типів шумів використовуватиметься функція **`imnoise2`** [2].

Відновлення зображень за присутності лише шумів у просторовій області

Якщо зображення спотворено виключно шумом, тобто:

$$g(x, y) = f(x, y) + \eta(x, y),$$

для його відновлення застосовуються методи, що зведені до табл. 4.1.

Табл. 3.1. Методи заглушення шумів в просторовій області

Назва фільтру	Формула
Арифметичне середнє	$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t)$
Геометричне середнє	$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}$
Гармонійне середнє	$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$
Контргармонійне середнє	$\hat{f}(x,y) = \sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1} \Big/ \sum_{(s,t)\in S_{xy}} g(s,t)^{Q}$
Медіана	$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$
Максимум	$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}} \{g(s,t)\}$
Мінімум	$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}} \{g(s,t)\}$
Середня точка	$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}} \{g(s,t)\} + \min_{(s,t)\in S_{xy}} \{g(s,t)\} \right]$
α-усічене середнє	$\hat{f}(x,y) = \frac{1}{mn-\alpha} \sum_{(s,t)\in S_{xy}} g(s,t)$

Для заглушення шумів в просторовій області також *використовується адаптивна медіанна фільтрація*, яка в порівнянні зі звичайним алгоритмом медіанної фільтрації використовує маски різного розміру. Це дозволяє в меншій мірі подавляти малі деталі на зображенні, роблячи його «розмитим», і ефективно заглушувати шуми. Алгоритм адаптивної медіанної фільтрації більш детально розглянуто на лекції, а функція яка його реалізує має назву **`admedian`**. Дана функція не є стандартною і надається окремо, її код можна знайти також в [2].

Відновлення зображень в частотній області

Якщо спотворюючий оператор відомий, то відновити зображення можливо наступним чином:

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}.$$

На базі даного принципу працює метод *інверсної фільтрації*, який представляють наступним рівнянням:

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}.$$

Іншим, більш ефективним, методом відновлення зіпсованого зображення є так звана *фільтрація Вінера*:

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \times \frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)} \right] G(u,v)$$

де $H(u, v)$ – спотворююча функція; $|H(u, v)|^2 = H^*(u, v) \cdot H(u, v)$; $H^*(u, v)$ – комплексно-спряжена функція до $H(u, v)$; $S_\eta(u, v) = |N(u, v)|^2$ – енергетичний спектр шуму; $S_f(u, v) = |F(u, v)|^2$ – спектр неспотвореного зображення; частка $S_\eta(u, v) / S_f(u, v)$ – енергетичне співвідношенням шум/сигнал (NSPR, Noise-to-Signal Power Ratio).

При застосуванні фільтрації Вінера, замість енергетичного співвідношення шум/сигнал застосовується величина:

$$R = \frac{\eta_A}{f_A},$$

в якій $\eta_A = \frac{1}{MN}\sum_u\sum_v S_\eta(u,v)$ – середня енергія шуму, $f_A = \frac{1}{MN}\sum_u\sum_v S_f(u,v)$ – середня енергія сигналу (зображення).

Для виконання відновлення зображень за допомогою інверсної фільтрації та фільтрації Вінера передбачено функцію **`deconvwnr`**. Дана функція приймає два для інверсної та три для вінерівскої фільтрації аргументи – спотворене зображення, спотворюючий оператор та співвідношення шум/сигнал *R*, відповідно.

Два попередні види фільтрації потребують знання спотворюючого оператора, однак на практиці це не є зручним. В такому випадку доцільно застосовувати методи *сліпої деконволюції*. Їх суть полягає в оцінюванні максимуму правдоподібності (MLE, Maximum-Likelihood Estimation) – стратегії оптимізації при побудові наближень величин, спотворених випадковим шумом. Коротко можна сказати, що в інтерпретації MLE зображення вважається випадково вибраним з деякою певною вірогідністю з сімейства інших можливих випадкових величин. Функція правдоподібності виражається через функції $g(x, y)$, $f(x, y)$ і $h(x, y)$, а задача полягає в знаходженні максимуму функції правдоподібності. При виконанні сліпої деконволюції процес оптимізації здійснюється ітераційно, за умови виконання відповідних обмежень та збіжності всієї процедури.

Для виконання сліпої деконволюції використовується функція **`deconvblind`**. Приклад її застосування розглянуто на лекції.

При виконанні обробки зображень у Python, можна скористатись функціями пакетів **`skimage.restoration`** та **`skimage.filters`**. Для створення тестового зображення у вигляді шахової дошки, можна використати функцію **`checkerboard`**, що надається з роботою.

Геометричні просторові перетворення

Геометричні спотворення зображення можна змоделювати (представити) за допомогою так званих *афінних перетворень*, які узагальнюються наступним матричним рівнянням:

$$[x \quad y \quad 1] = [w \quad z \quad 1]\mathbf{T} = [w \quad z \quad 1]\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix},$$

де x та y – координати результуючого зображення; w та z – координати вихідного зображення; **T** – матриця, що характеризує перетворення координат, а t_{ij} – її компоненти.

Визначаючи необхідним чином компоненти матриці **T** можна задати поворот, перенесення, розтягнення (масштабування) та зсув зображення. Для виконання геометричних перетворень, зокрема і афінних в MATLAB наявні наступні функції:

maketform створює по заданому типу перетворення та матриці **T** структуру, за допомогою якої виконується просторове перетворення зображення.

cp2tform генерує структуру для здійснення геометричних перетворень зображення на основі пар контрольних точок, які задають куди мають бути перенесені координати вихідного зображення для отримання бажаного результату.

imtransform виконує геометричне перетворення заданого зображення, користуючись структурами, що створюються попередніми функціями.

cpselect GUI інтерфейс для ручного вибору контрольних точок на парі зображень.

У Python для аналогічних операцій можна скористатись функціями **AffineTransform**, **ProjectiveTransform**, **warp** та **estimate** пакету **skimage.transform**.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові m-файли чи ру-файли з функціями надаються окремо).
2. Дослідити вплив параметрів використовуваних процедур на результат обробки.
3. Представити процедури обробки зображень у вигляді скрипта на мові MATLAB чи Python.

Завдання

Виконати наступні операції:

1. Генерування шуму із заданим розподілом.
2. Придушення періодичного шуму (файл – pic.0.jpg).
3. Аналіз параметрів шуму (файл – pic.1.jpg).
4. Заглушення шумів за допомогою просторової фільтрації (файл – pic.2.jpg та pic.3.jpg).
5. Заглушення шумів за допомогою адаптивної медіанної фільтрації (файл – pic.4.jpg).
6. Моделювання спотворюючої функції.
7. Інверсну фільтрацію зображення.
8. Вінерівську фільтрацію зображення.
9. Сліпу деконволюцію.
10. Геометричні просторові перетворення зображення.
11. Реєстрація зображень (файл – pic.5.jpg та pic.6.jpg).

Запитання для самоконтролю

1. Як можна змоделювати процес спотворення зображення.
2. Наведіть відомі Вам моделі шуму.
3. Як виконується фільтрація/подавлення шуму.
4. В чому суть адаптивної медіанної фільтрації.

5. Які методи фільтрації використовуються для виправлення спотворень.
6. Які недоліки інверсної фільтрації.
7. В чому полягають переваги вінерівської фільтрації в порівнянні з інверсною. Який основний спільний недолік має вінерівська та інверсна фільтрації.
8. Як моделюються геометричні просторові перетворення.
9. Які типи геометричних просторових перетворень Вам відомі. Коли вони застосовуються та які типи спотворень здатні виправляти.

ЛАБОРАТОРНА РОБОТА №4

Обробка кольорових зображень

Тема роботи: обробка кольорових зображень.

Мета роботи: дослідити особливості обробки кольорових зображень в середовищі MATLAB або Python.

Теоретичні відомості

Основи теорії кольору

Людиною колір сприймається за характером відбитого від об'єктів або випромінюваного ними світла. Видиме світло складає відносно вузьку смугу всього діапазону довжин хвиль електромагнітного спектру: від 380 до 780 нм. Тіло, що рівномірно відбиває або випромінює світло у всьому видимому діапазоні довжин хвиль, для спостерігача має вигляд білого. В той же час тіло, яке відбиває світло переважно в деякому обмеженому діапазоні довжин хвиль, набуває певного кольору.

Коли світло є безбарвним (*ахроматичним*) в ролі його характеристики виступає інтенсивність. Параметрами, якими прийнято користуватись для розрізнення кольорів є:

- *Світлість* (*яскравість*) – характеризує інтенсивність світла.
- *Тон кольору* – характеризує домінуючий колір, що сприймається спостерігачем і зазвичай відповідає певному спектральному складу світла.
- *Насиченість* – характеризує білизну кольору.

Разом тон кольору та насиченість називаються *кольоровістю*.

Кольорові простори

Кольоровий простір або кольорова модель визначають певну систему координат та підпростір у цій системі, в якому кожному кольору відповідає єдина точка. Існують наступні види просторів кольору:

- Лінійні (XYZ, RGB, CMY[K]).
- Нелінійні (HSV, HSI);
- Рівномірні (L*a*b*, L*u*v*).

Для перетворення кольорів в системі MATLAB використовуються наступні функції:

- **`rgb2gray`**
- **`rgb2hsv`** та **`hsv2rgb`**
- **`rgb2xyz`** та **`xyz2rgb`**
- **`rgb2lab`** та **`lab2rgb`**
- тощо

або пари універсальних функції:

- **`makecform`**
- **`applycform`**

Перетворення кольорів в середовищі Python можна здійснити за допомогою аналогічних функцій пакету **`skimage.color`**. При цьому у Python замість пари функцій **`makecform`** та **`applycform`** застосовується єдина функція **`convert_colorspace`**.

Обробка кольорових зображень

Фільтрацію цифрових кольорових зображень можна здійснювати покомпонентно, тобто по кожній кольоровій площині (складовій) окремо. Для цього можна застосовувати *деякі* розглянуті вище методи просторової та частотної обробки (див. лаб. 2 та лаб. 3).

Водночас, підвищення різкості та згладжування можна досягти шляхом обробки компоненти V моделі кольору HSV або компоненти L* моделей L*u*v* та L*a*b*. В такому разі алгоритм фільтрації матиме вигляд:

1. Перевести зображення із заданого простору в один із просторів HSV/L*u*v*/L*a*b*.
2. Виконати фільтрацію компоненти V або L*.
3. Повернути зображення у заданий кольоровий простір.

В ряді випадків, покомпонентна обробка зображень може призводити до кольорових спотворень. Цей ефект виникатиме тоді, коли оператори, застосовувані до площин кольору зображення відрізнятимуться, зокрема при покомпонентній еквалізації гістограм. В такому випадку застосовують перетворення в простори HSV, L*u*v* та L*a*b*.

Зауваження. При виконанні перетворень між кольоровим просторами необхідно контролювати типи даних утворюваних зображень, щоб при подальшій обробці випадково не вийти за діапазон допустимих значень пікселів, що може призводити до появи спотворень.

Виявлення контурів на кольорових зображеннях

Наближено знайти градієнт кольорового RGB зображення можна знайти шляхом покомпонентної обробки наступним чином:

$$F_a(x,y) = \nabla f_{\mathrm{R}} + \nabla f_{\mathrm{G}} + \nabla f_{\mathrm{B}} = \\ = \sqrt{\left(\frac{\partial \mathrm{R}}{\partial x}\right)^2 + \left(\frac{\partial \mathrm{R}}{\partial y}\right)^2} + \sqrt{\left(\frac{\partial \mathrm{G}}{\partial x}\right)^2 + \left(\frac{\partial \mathrm{G}}{\partial y}\right)^2} + \sqrt{\left(\frac{\partial \mathrm{B}}{\partial x}\right)^2 + \left(\frac{\partial \mathrm{B}}{\partial y}\right)^2}$$

де частинні похідні визначаються будь-яким розглянутим вище методом диференціювання зображень, наприклад, за допомогою оператора Собела; R, G та B – відповідні зображення-компоненти.

Водночас існує більш точний метод виявлення контурів, що оснований на довизначенні поняття градієнту для вектор-функцій, якими власне і є кольорові зображення. Нехай існують наступні величини g_{xx}, g_{yy} та g_{xy}:

$$g_{xx} = \mathbf{u}^2 = \mathbf{u}^T\mathbf{u} = \left|\frac{\partial \mathrm{R}}{\partial x}\right|^2 + \left|\frac{\partial \mathrm{G}}{\partial x}\right|^2 + \left|\frac{\partial \mathrm{B}}{\partial x}\right|^2,$$

$$g_{xx} = \mathbf{v}^2 = \mathbf{v}^T\mathbf{v} = \left|\frac{\partial \mathrm{R}}{\partial y}\right|^2 + \left|\frac{\partial \mathrm{G}}{\partial y}\right|^2 + \left|\frac{\partial \mathrm{B}}{\partial y}\right|^2,$$

$$g_{xy} = \mathbf{u}\cdot\mathbf{v} = \mathbf{u}^T\mathbf{v} = \frac{\partial \mathrm{R}}{\partial x}\frac{\partial \mathrm{R}}{\partial y} + \frac{\partial \mathrm{G}}{\partial x}\frac{\partial \mathrm{G}}{\partial y} + \frac{\partial \mathrm{B}}{\partial x}\frac{\partial \mathrm{B}}{\partial y}.$$

Можна, тоді показати, що кут, в напрямку якого швидкість росту вектор-функції буде максимальною, визначається рівнянням:

$$\mathrm{tg}(2\theta) = \frac{2g_{xy}}{g_{xx} - g_{yy}},$$

а величина швидкості зміни в точці (x, y) в напрямку кута θ задовольняє виразу:

$$F(x, y) = \sqrt{\tfrac{1}{2}[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})\cos(2\theta) + g_{xy}\sin(2\theta)]}.$$

Відзначимо, що наведене вище рівняння для кута θ має два розв'язки, які відрізняються на 90°, тому щоб знайти градієнт необхідно обчислити $F(x, y)$ для двох значень кутів: θ та $\theta + \pi / 2$ та вибрати з них більше.

Для виявлення контурів на кольорових зображеннях використовуватимемо функцію **`colorgrad`**, яка реалізує обидва вище описаних методи.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові скрипти з функціями надаються окремо).
2. Провести експериментальні дослідження впливу параметрів функцій обробки на якість результуючих зображень.
3. На практиці з'ясувати що є спільним, а що відмінним при обробці кольорових та яскравісних зображень.
4. Представити процедури обробки зображень у вигляді файла-скрипта.

Завдання

Виконати обробку заданих зображень:

1. Перетворити зображення у різні простори кольору (файл – pic.1.png).
2. Здійснити згладжування зображення за допомогою покомпонентної обробки (файл – pic.2.png).
3. Здійснити згладжування зображення шляхом його перетворення в системи кольору HSV та L*a*b* (файл – pic.2.png).

4. Підвищити різкість зображення за допомогою покомпонентної обробки (файл – pic.2.png).
5. Підвищити різкість зображення шляхом його перетворення в системи кольору HSV та L*a*b* (файл – pic.2.png).
6. Виконати еквалізацію гістограм (файл – pic.2.png).
7. Виконати виявлення контурів на зображенні покомпонентно та розглядаючи зображення як вектор функцію (файл – pic.2.png). Порівняти результати.

Запитання для самоконтролю

1. Що таке колір, якими параметрами він зазвичай характеризується.
2. Як представляються кольорові цифрові зображення у сучасній обчислювальній техніці.
3. Що таке кольоровий простір. Наведіть види кольорових просторів, та їх приклади.
4. Для чого потрібні різні кольорові простори. Наведіть приклади їх застосування.
5. Якими двома шляхами можна виконувати обробку кольорових зображень. В чому їх переваги та недоліки.
6. Як виконується еквалізація гістограм при обробці кольорових зображень. Чому її не можна виконувати по кожній площині кольору.
7. Наведіть алгоритм обробки зображення із переходом у простори HSV/L*u*v/L*a*b.
8. Як виконується виявлення контурів на кольорових зображеннях.
9. Які процедура переходу між кольоровими просторами застосовуються в системі MATLAB.

ЛАБОРАТОРНА РОБОТА №5

Морфологічна обробка зображень

Тема роботи: морфологічна обробка зображень.

Мета роботи: дослідити та застосувати основні методи морфологічної обробки зображень в середовищі MATLAB або Python.

Теоретичні відомості

Базові поняття теорії множин, що застосовуються в морфологічній обробці зображень

Базові поняття теорії множин, які застосовуються в морфологічній обробці зображень проілюстровано нижче на рис. 5.1

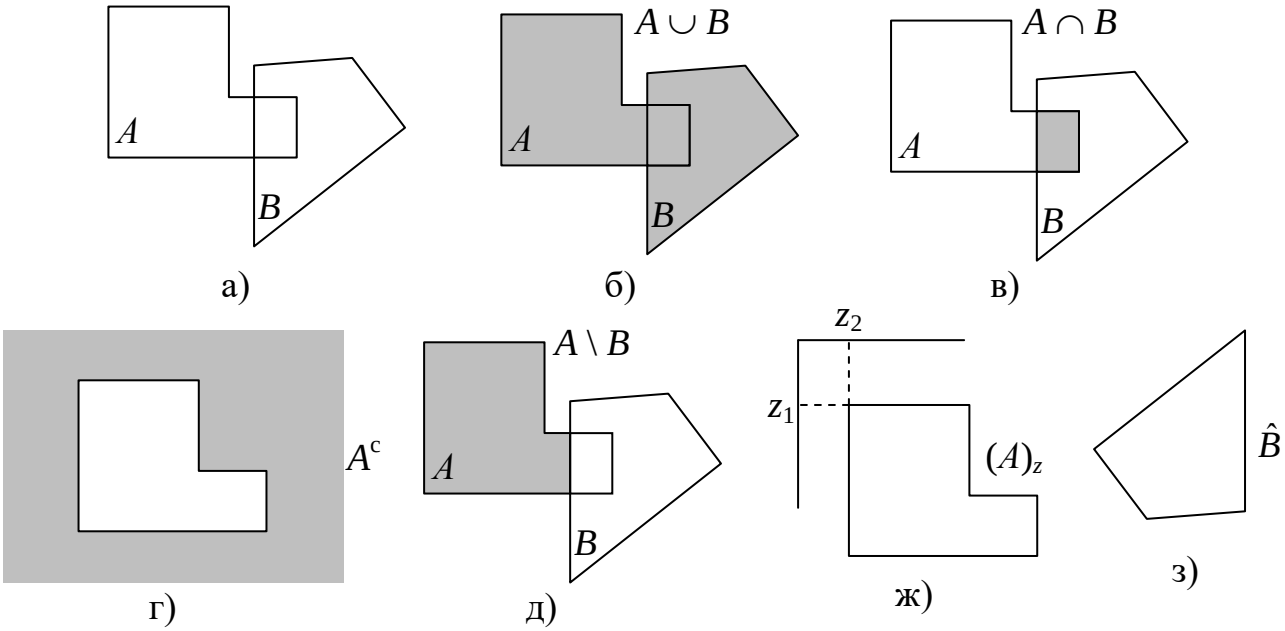

Рис. 5.1. Базові поняття теорії множин:

а) дві множини A та B;

б) об'єднання множин A та B: $C = A \cup B$;

в) перетин множин A та B: $C = A \cap B$;

г) доповнення множини A: $A^c = \{w \mid w \notin A\}$;

д) різниця множин A та B: $A \setminus B = \{w \mid w \in A, w \notin B\}$;

ж) зсув множини A: $(A)_z = \{c \mid c = a + z, a \in A\}$;

з) центральне віддзеркалення множини B: $\hat{B} = \{w \mid w = -b, b \in B\}$.

Дилатація та ерозія

Нехай A та B – множини з простору Z^2.

Дилатація множини A по множині B (або відносно B) позначається $A \oplus B$ і визначається як:

$$A \oplus B = \{z \mid (B)_z \cap A \neq \varnothing\}.$$

Дилатація множини A по B – це множина всіх таких зсувів z, за яких множини B і A збігаються щонайменше в одному елементі.

Ерозія A по B, що позначається $A \ominus B$, визначається як

$$A \ominus B = \{z \mid (B)_z \cap A^c = \varnothing\}.$$

Інакше кажучи, ерозія множини A по примітиву B – це множина всіх таких точок z, при зсуві в які множина B цілком міститься в A.

Півтонова дилатація зображення f по структуроутворюючому елементу b позначається $f \oplus b$ і визначається формулою:

$$(f \oplus b)(x, y) = \max\{f(x - x', y - y') + b(x', y') \mid (x', y') \in D_b\},$$

де D_b позначає область визначення b, і передбачається, що $f(x, y)$ дорівнює $-\infty$ поза областю визначення f. Це рівняння реалізує процес, аналогічний процедурі знаходження просторової згортки.

Півтонова ерозія зображення f по структуроутворюючому елементу b позначається $f \ominus b$ і визначається як:

$$(f \ominus b)(x, y) = \min\{f(x - x', y - y') - b(x', y') \mid (x', y') \in D_b\},$$

де D_b позначає область визначення b, і передбачається, що $f(x, y)$ дорівнює $+\infty$ поза областю визначення f.

Процедури ерозії та дилатації в системі MATLAB реалізовані функціями **`imerode`** та **`imdilate`** відповідно. Вони виконують звичайні чи півтонові ерозію та дилатацію, в залежності від класу (типу даних) аргументу.

У Python для виконання морфологічних операцій рекомендовано використовувати пакет **`skimage.morphology`**. В даному пакеті ерозія та

дилатація виконуються функціями **`binary_erosion`**, **`binary_dilation`** для двійкових зображень та **`erosion`** і **`dilation`** для півтонових зображень.

Замикання та розмикання

Розмикання множини A по примітиву B позначається $A \circ B$ і визначається рівністю:

$$A \circ B = (A \ominus B) \oplus B.$$

Таким чином, розмикання множини A по примітиву B будується як ерозія A по B, результат якої потім піддається дилатації по тому ж примітиву B.

З*амикання* множини A по примітиву B позначається як $A \bullet B$ і визначається наступним чином:

$$A \bullet B = (A \oplus B) \ominus B,$$

тобто, як дилатація A по B, результат якої потім піддається ерозії по тому ж примітиву B.

Півтонові операції замикання та розмикання аналогічні звичайним відповідним їм операціям, за виключенням того, що тут застосовуються півтонові ерозія та дилатація.

Процедури розмикання та замикання в системі MATLAB реалізовані функціями **`imopen`** та **`imclose`** відповідно. Вони виконують звичайні чи півтонові розмикання та замикання, в залежності від класу (типу даних) аргументу. У Python розмикання та замикання зображень реалізовані функціями **`binary_opening`** і **`opening`** та **`binary_closing`** і **`closing`** відповідно.

Перетворення успіх невдача

Перетворення успіх/*невдача* множини A по множині B позначається через $A \otimes B$. Тут B позначає структуротворну пару елементів $B = (B_1, B_2)$, а не один елемент, як це було раніше. За визначенням, перетворенням множини A по B називається множина:

$$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2).$$

Аналіз компонент зв’язності

Піксел p з координатами (x, y) може мати 4 або 8 сусідніх пікселів, які позначаються як $N_4(p)$, $N_D(p)$ та $N_8(p)$ і показані сірим кольором на рис. 5.2.

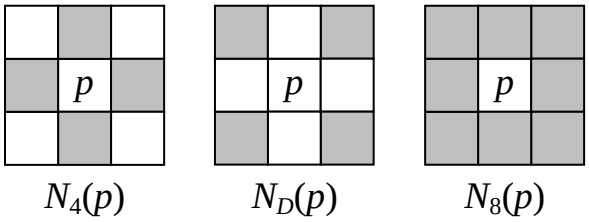

Рис. 5.2. Сусіди піксела p та їх позначення

Шляхом між пікселами p_1 і p_n називається послідовність пікселів $p_1, p_2, \ldots, p_{n-1}, p_n$, така, що p_k є суміжним для p_{k+1} при $k = 1, 2, \ldots, n - 1$. Шлях може бути *4-зв’язним* або *8-зв’язним*, залежно від використовуваного визначення суміжності пікселів.

Нехай S – деяка підмножина елементів зображення. Два його елементи p та q називаються зв’язаними в S, якщо між ними існує шлях, що цілком складається з елементів підмножини S. Для будь-якого піксела p із S множина всіх пікселів, зв’язаних з ним в S називається *компонентою зв’язності S*. Якщо множина S містить тільки одну компоненту зв’язності, то вона називається *зв’язною множиною*.

Для визначення компонент зв’язності в системі MATLAB використовуються функції **`bwconncomp`**, **`bwlabel`**. Для аналізу параметрів компонент зв’язності існує функція **`regionprops`**. Пошук компонент зв’язності у Python можна провести за допомогою функції **`label`** пакету **`skimage.morphology`**, а їх аналіз – за допомогою однойменної функції **`regionprops`** пакету **`skimage.measure`**.

Морфологічна реконструкція

Реконструкція є морфологічним перетворенням, в якому беруть участь два зображення і один структуроутворюючий елемент. Одне зображення, яке називається *маркером*, є вихідною точкою перетворення. Інше

зображення, *маска,* накладає певні обмеження (зв'язки) на відображення. Використовуваний структуроутворюючий елемент визначає зв'язність.

Якщо g – це маска, а f – маркер, то реконструкція g по f, яка позначається $R_g(f)$, визначається наступною ітеративною процедурою.

1. Ініціалізація: привласнити h_1 маркерне зображення f.
2. Побудувати структуроутворюючий елемент `B = ones(3)`.
3. Повторювати: $h_{k+1} = (h_k \oplus B) \cap g$ до тих пір, поки не стане $h_{k+1} = h_k$.

Маркер f має бути підмножиною g, тобто $f \subseteq g$.

Процедура морфологічної реконструкції в системі MATLAB реалізована функцією **`imreconstruct`**. У Python аналогічна операція може бути виконана функцією **`reconstruction`** пакету **`skimage.morphology`**.

Інші поширені морфологічні операції

Серед інших поширених морфологічних операцій можна виділити:

- Потоншення
- Потовщення
- Побудову остова (скелетонізація)
- Заповнення порожнистих зв'язних областей
- Сполучення близько розташованих зв'язних областей.
- Операція «виступ» (tophat).
- Операція «впадина» (bottomhat).
- Отримання контуру зв'язної області.

Для виконання зазначених операцій у MATLAB використовується функція **`bwmorph`**. У Python ці операції виконуються окремими функціями наявними у пакеті **`skimage.morphology`**.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові процедури надаються окремо).

2. Провести експериментальні дослідження впливу параметрів функцій обробки на якість результуючих зображень.
3. Представити процедури обробки зображень у вигляді файла-скрипта.

Завдання

Виконати обробку заданих зображень:

1. Дилатація (файл – pic.1.jpg).
2. Ерозія (файл – pic.2.jpg).
3. Розмикання та замикання (файл – pic.3.jpg).
4. Потоншення (файл – pic.3.jpg).
5. Побудова остова (файл – pic.4.jpg).
6. Виділення компонент зв’язності (файл – pic.5.jpg).
7. Морфологічна реконструкція (слайд – 14, файли – pic.6a.tif та pic.6b.tif).
8. Морфологічна реконструкція (слайд – 15, файл – pic.7.tif).
9. Півтонова дилатація та ерозія (файл – pic.8.tif).
10. Півтонове розмикання та замикання (файл – pic.9.tif).
11. Морфологічний градієнт (файл – cameraman.tif).
12. Перетворення «виступ» (файл – rice.png).
13. Морфологічна півтонова реконструкція (файл – pic.10.tif).

Запитання для самоконтролю

1. В чому полягає суть операцій ерозії та дилатації для бінарних зображень. Як їх можна застосовувати для аналізу зображень.
2. В чому полягає відмінність операцій ерозії та дилатації для півтонових зображень. Для чого застосовуються ці операції.
3. Що таке операції розмикання та замикання. Які дії із зображеннями дозволяють виконувати дані операції.
4. Що таке перетворення успіх/невдача, для чого це перетворення може застосовуватись.
5. Що таке зв’язність, якою вона буває.

6. Дайте визначення компоненту зв’язності.
7. Для чого застосовується морфологічна реконструкція.
8. Наведіть алгоритм морфологічної реконструкції. Який аналіз зображень дозволяє даний алгоритм виконувати.
9. На прикладі функцій **`regionprops`** та **`bwmorph`**, наведіть які ще види аналізу зображень можна виконувати за допомогою морфологічних операцій.

ЛАБОРАТОРНА РОБОТА №6

Сегментація зображень

Тема роботи: методи сегментації зображень.

Мета роботи: дослідити методи сегментації зображень в середовищі MATLAB або Python.

Теоретичні відомості

Виявлення точок

Для виявлення точок на фрагментах зображень, на яких яскравість змінюється повільно доцільно використовувати похідні, зокрема лапласіан. Піксел, в якому є точка (неоднорідність яскравості) можна виявити наступним чином:

$$|R| \geq T$$

де T – ненегативний поріг (зазвичай $T \sim \max(|R|)$), а R – похідні зображення.

Виявлення ліній

Виявлення ліній також здійснюється шляхом застосування до зображення похідних. При цьому, маски похідних повинні мати особливий вигляд: рис. 6.1

–1	–1	–1
2	2	2
–1	–1	–1

Горизонтальна

–1	–1	2
–1	2	–1
2	–1	–1

+45°

–1	2	–1
–1	2	–1
–1	2	–1

Вертикальна

2	–1	–1
–1	2	–1
–1	–1	2

–45°

Рис. 6.1. Маски для виявлення ліній

При застосування до зображення першої маски, найбільш сильний відгук спостерігатиметься на горизонтальних лініях завтовшки в один піксел. Аналогічно, друга маска дає найбільший відгук на лініях, що

проходять під кутом +45°; третя – на вертикальних лініях; четверта – на тих, що проходять під кутом –45°.

Виявлення перепадів

Для виявлення перепадів яскравості на зображенні, як і для виявлення точок та ліній застосовуються похідні першого й другого порядків. Виявлення перепадів в MATLAB можна виконати за допомогою спеціальної функції **`edge`**. У Python можна для цих цілей можна використати метод Кенні з пакету **`skimage.feature`**.

Поргова обробка

Порогове перетворення може розглядуватися як операція, що передбачає порівняння елементів зображення з функцією *T,* яка має вигляд:

$$T = T(x, y, p(x, y), f),$$

де f – зображення, а $p(x, y)$ позначає деяку локальну характеристику точки (x, y) зображення, наприклад, середню яскравість в околиці з центром в цій точці.

Якщо значення T залежить тільки від f, тобто однаково для всіх точок зображення, то такий поріг називається *глобальним.* Якщо поріг T залежить від просторових координат x та $y,$ то він називається *локальним* або *динамічним.*

Якщо T залежить від $p(x, y)$, то такий поріг називається *адаптивним.*

Автоматичний вибір порогу можна виконати наступною інтеративною процедурою:

1. Вибрати деяку початкову оцінку для значення порогу $T.$
2. Зробити сегментацію за допомогою порогу T (утворюються дві групи пікселів: G_1 и G_2).
3. Обчислити середню яскравість пікселів μ_1 та μ_2 в областях G_1 та $G_2.$
4. Обчислити нове значення порогу:

$$T = 0{,}5(\mu_1 + \mu_2).$$

5. Повторювати кроки з 2-го по 4-й до тих пір, поки різниця порогів T для сусідніх ітерацій не стане менше наперед заданого значення T_0.

Дана процедура реалізована в стандартній функції MATLAB і називається **`graythresh`**. Для виконання адаптивної порогової обробки перед операцію **`graythresh`** можна застосувати перетворення «*виступ*» (див. лаб. 6).

Сегментація перетворенням вододілів

Щоб зрозуміти перетворення *вододілу*, необхідно уявити собі півтонове зображення у вигляді поверхні рівнів, на якій величини $f(x, y)$ інтерпретуються як висоти. Якщо уявити собі дощ, що йде над цією поверхнею, то, очевидно, вода збиратиметься в областях, помічених як водозбірні басейни. Краплі дощу, падаючі точно на лінію, помічену як лінія вододілу, з однаковою імовірністю можуть потрапляти як в один, так і в інший басейн.

Перетворення вододілу знаходить водозбірні басейни і будує лінію вододілу на півтоновому зображенні. Якщо використовувати термінологію сегментації, то ключовим моментом є зміна початкового зображення і перетворення його в таке зображення, що водозбірними басейнами на ньому були області, що відповідають об'єктам, які ми хочемо сегментувати.

Для сегментації зображення за допомогою перетворення вододілу в MATLAB передбачена функція **`watershed`**. У Python пакет **`skimage.morphology`** має функцію з аналогічним іменем, хоча вона працює дещо інакше, ніж функція MATLAB (див. документацію і приклади).

З метою підвищення якості сегментації двійкових зображень за допомогою перетворення вододілу доцільно використовувати *перетворення відстані* двійкового зображення, яке є досить простою функцією: воно дорівнює відстані від кожного піксела до найближчого піксела з ненульовим значенням. Перетворення відстані в MATLAB можна здійснити

за допомогою функції **`bwdist`**. У Python перетворення відстані може бути виконане функцією **`distance_transform_edt`** пакету **`scipy.ndimage`**.

Модуль градієнта часто використовується для попередної обробки півтонових зображень перед сегментацією по вододілах. Піксели зображення-градієнта з великими значеннями розташовуються поблизу кордонів об'єктів. Таким чином, застосовуючи перетворення вододілу до модуля градієнта, можна отримати лінії вододілів уздовж кордонів об'єктів.

Ще одним підходом, що дозволяє підвищити якість сегментації, зменшуючи надлишкову сегментацією, оснований на *ідеї маркерів*. *Маркер* є зв'язною компонентою, що належить зображенню. Розрізнятимемо *внутрішні маркери,* що належать до об'єктів, які нас цікавлять, і *зовнішні маркери* – ті що відповідають фону зображення. Суть методу полягає в обмеженні за допомогою маркерів розмірів сегментів. Детальніше див. лекцію.

Сегментація шляхом кластеризації за *k*-середніми

Суть алгоритму кластеризації по *k*-середніх полягає в оптимізації цільової функції, яка відображає якість розбиття на кластери (сегменти) заданого набору даних (зображення). Цільову функцію отримують із припущення, що існує *k* кластерів, де *k* є відомим числом, і вважаючи, що конкретний елемент даних розташовується поблизу центру відповідного йому кластера. Таким чином, у ролі цільової функції виступає деяка міра відстані, наприклад, *евклідова*:

$$\Phi = \sum_{i=1}^{k} \left[\sum_{j \in \mathbf{I}_i} (x_j - c_i)^T (x_j - c_i) \right],$$

де k – кількість кластерів; $\mathbf{I}_i$ – множина індексів елементів, що належать до i-го кластеру; x_j – j-а інформаційна точка; c_i – центр i-го кластеру.

Алгоритм сегментації по *k*-середніх можна описати наступною послідовністю дій:

1. Обрати *k* інформаційних точок у якості центрів кластерів.

2. Віднести кожну інформаційну точку до того кластера, відстань до якого від даної точки є найменшою.
3. Впевнитися, що кожен кластер містить хоча б одну точку. Для цього кожен пустий кластер може бути доповнений довільною інформаційною точкою, розташованою далеко від його центру.
4. Центр кожного кластера замінити середнім значенням елементів, що йому відповідають.
5. Повторювати кроки 2-4, доки не припиниться зміна центрів кластерів.

Для виконання сегментації по *k*-середніх можна скористатись функцією **`kmeans`**, яка входить в стандартний набір функцій MATLAB. Для виконання сегментації за *k*-середніми у Python найпростіше скористатися функцією **`kmeans`** пакету **`cv2`** (OpenCV). Особливості використання даної функції описані в документації.

Порядок виконання роботи

1. У відповідністю до наведених нижче завдань виконати обробку зображень (зображення та необхідні для їх обробки додаткові файли з функціями надаються окремо).
2. Дослідити як параметри функцій обробки впливають на отримуваний результат.
3. Представити процедури обробки зображень у вигляді файла-скрипта.

Завдання

Виконати обробку заданих зображень:

1. Виявити точки на зображенні (файл – pic.1.tif);
2. Виявити ліній на зображенні (файл – pic.2.tif);
3. Виявити перепади яскравості на зображенні (файл – pic.3.tif);
4. Обробка з глобальним порогом (файл – pic.4.tif);

5. Сегментація по вододілам за допомогою перетворення відстані (файл – pic.5.tif);
6. Сегментація по вододілам за допомогою градієнтів (файл – pic.6.tif);
7. Виконати сегментацію кольорового зображення за допомогою кластеризації по *k*-середніх (файл – pic.8.jpg).

Запитання для самоконтролю

1. Як здійснюється виділення точок, ліній.
2. Які методи застосовуються для виявлення перепадів яскравості на зображеннях.
3. Що таке порогова обробка (порогове перетворення) зображень, як її формально можна представити.
4. Наведіть процедуру порогової обробки зображень із глобальним порогом.
5. Як можна виконати обробку зображень із адаптивним порогом.
6. В чому полягає суть перетворення вододілу. Як із допомогою цього перетворення можна виконувати сегментацію.
7. Які підходи використовуються для підвищення надійності сегментації за вододілами.
8. Наведіть алгоритм сегментації зображень за *k*-середніми.

ЛАБОРАТОРНА РОБОТА №7

Розпізнавання образів

Тема роботи: методи розпізнавання зображень.

Мета роботи: дослідити та реалізувати методи розпізнавання зображень.

Теоретичні відомості

Образом називається деяка впорядкована сукупність *дескрипторів*, які в літературі по розпізнаванню часто називають *ознаками*. *Класом образів* (або просто *класом*) називається сукупність образів, що володіють деякими спільними рисами (властивостями) – однаковими ознаками.

Набули поширення три форми впорядкованого представлення ознак. Зокрема у вигляді:

- векторів ознак (для кількісних дескрипторів);
- символьних рядків;
- дерев.

На практиці задача класифікації найчастіше зводиться до побудови деякої гіперплощини або множини гіперплощин в просторі ознак, щоб з їх допомогою можливо було визначити якому класу належить задана множина ознак.

Класифікатор за мінімумом відстані

Прототип кожного класу визначається як вектор математичного очікування образів даного класу:

$$\mathbf{m}_j = \frac{1}{N} \sum_{x \in w_j} \mathbf{x}_j,\ j = 1,2,...,W$$

де $\mathbf{x}_j$ – вектори ознак образів j-го класу, N – кількість образів, W – кількість класів. Підсумовування ведеться по всіх таких векторах ознак об’єктів класу

j. Розділююча поверхня між класами ω_1, і ω_2 у випадку використання класифікатора за мінімумом відстані задається рівнянням:

$$d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i - \mathbf{m}_j) = 0.$$

Кореляційне співставлення

Для пошуку об'єкта на зображенні часто використовується кореляція. В безпосередньому вигляді її застосовують доволі рідко, оскільки вона чутлива до зміни яскравості порівнюваних зображень і з огляду на це не дозволяє досягти достатньої надійності. Натомість більшого розповсюдження набув коефіцієнт кореляції, що обчислюється за формулою:

$$\gamma(x,y) = \frac{\sum_s \sum_t [f(s,t) - \overline{f}(s,t)] \cdot [w(x+s, y+t) - \overline{w}]}{\sqrt{\sum_s \sum_t [f(s,t) - \overline{f}(s,t)]^2 \cdot \sum_s \sum_t [w(x+s, y+t) - \overline{w}]^2}},$$

де $x = 0, 1, 2, ..., M-1$, $y = 0, 1, 2, ..., N-1$, $\overline{w}$ – середнє значення пікселів шаблону w (обчислюване тільки один раз), $\overline{f}$ – середнє значення елементів зображення f в області, що збігається з поточним положенням w, а підсумовування ведеться за всіма парам координат, спільним для f та w. Коефіцієнт кореляції змінюється в діапазоні $\gamma(x,y) \in [-1;+1]$ і не залежить від зміни масштабу яскравостей f та w.

В системі MATLAB коефіцієнт кореляції між двомірними даними (зображеннями) обчислюється за допомогою функції **`normxcorr2`**. При використанні Python можна скористатись функцією **`matchTemplate`** пакету OpenCV або функцією **`match_template`** пакету **`skimage.feature`**.

Штучні нейронні мережі

Елементарною коміркою нейронної мережі є *нейрон.* Структура простого нейрона показана на рис. 7.1. Він має вектор входу **p**, що містить R елементів $p_1, p_2, ..., p_R$. Кожен вхідний елемент перемножується на

відповідні вагові коефіцієнти $w_1, w_2, ..., w_R$, після чого отримані добутки додаються. Таким чином, якщо розглядати вагові коефіцієнти як вектор-рядок **W**, то результат підсумовування зважених вхідних сигналів є не чим іншим, як скалярним добутком між векторами **W** та **p**.

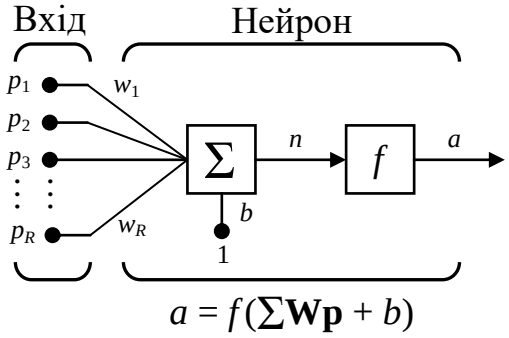

Рис. 7.1. Нейрон з векторним входом

Нейрон має зміщення *b*, яке підсумовується разом із зваженою сумою входів. Таким чином, сума *n* набуває вигляду $n = w_1 p_1 + w_2 p_2 + ... + w_R p_R + b$ і є рівнянням гіперплощини в *R*-мірному просторі, причому ваги визначатимуть нахил даної гіперплощини, а зміщення – її положення. Сума *n* передається як аргумент до *функції активації f*, яка визначає кінцеву реакцію нейрона. Найбільш поширені функції активації наведені на рис. 7.2. Відмітимо, що в сучасних мережах у проміжних шарах найбільш часто використовуються функції типу ReLU д) та е), а у вихідному шарі функція для отримання *категорійного розподілу* softmax.

Реальна нейронна мережа може містити один або більшу кількість шарів. Одношарові нейронні мережі мають суттєве обмеження – вони можуть виконувати розпізнавання *тільки лінійно-роздільних образів*, що звужує їх застосування. Тому в більшості реальних задач розповсюдження набули багатошарові нейронні мережі, одна з яких показана на рис. 7.3. Ця мережа має *R* входів, S^1 нейронів в першому шарі, S^2 нейронів в другому шарі і так далі. При цьому різні шари можуть мати різне число нейронів. В*и*ходи кожного проміжного шару виступають входами для наступного шару. Матриці ваг вхідного шару і прихованих (проміжних) шарів позначено **IW** (Input Weight) і **LW** (Layer Weight) відповідно.

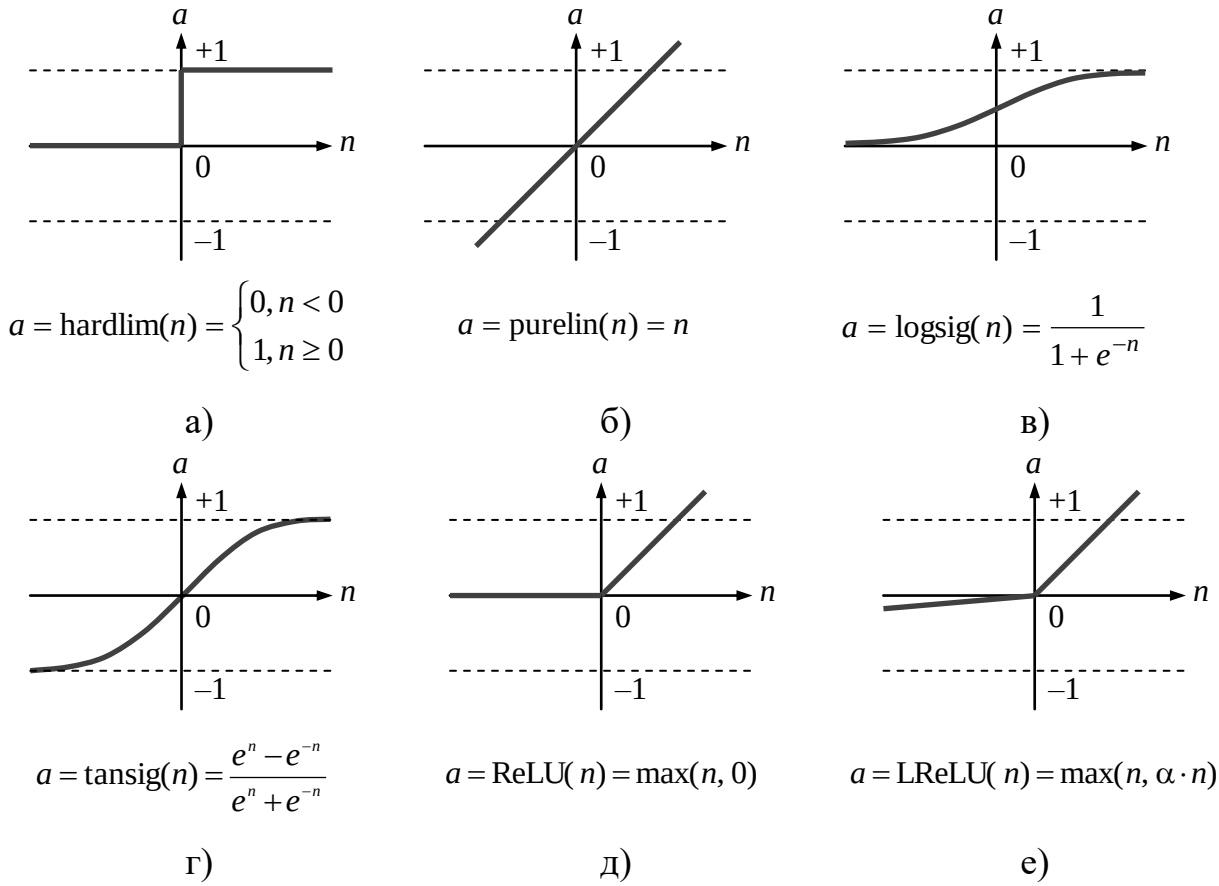

Рис. 7.2. Найбільш поширені функції активації:
а) функція Хевісайда;
б) лінійна функція активації;
в) логістична функція активації (сигмоїд);
г) тангенс гіперболічний;
д) функція ReLU (Rectified Linear Unit);
е) функція Leaky ReLU (LReLU)

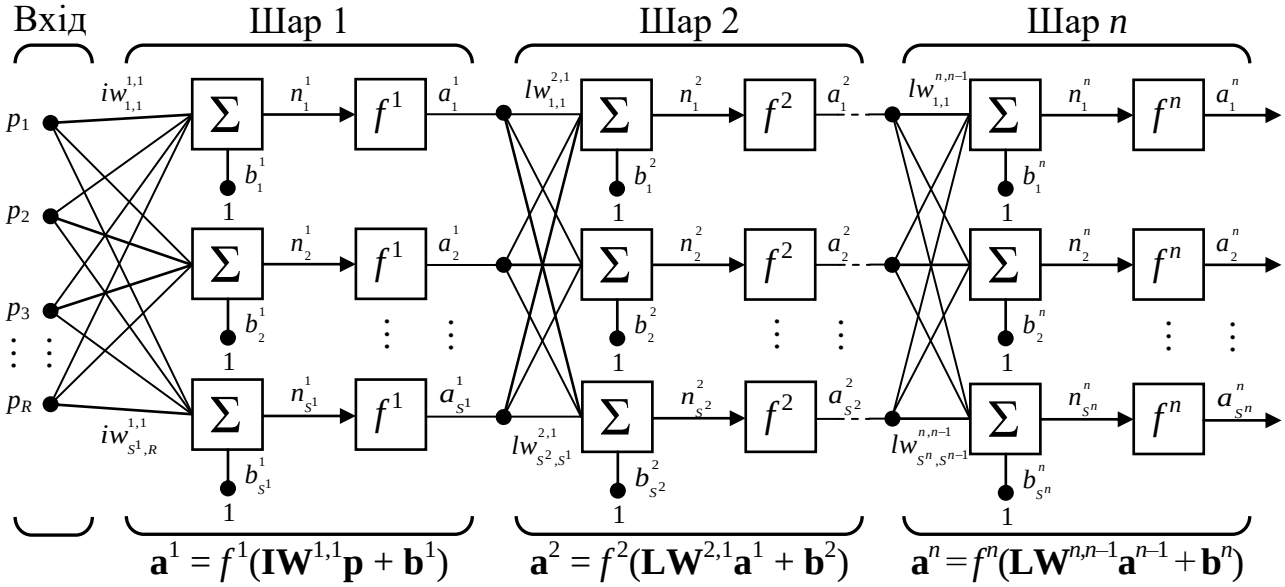

Рис. 7.3. Повнозв’язна багатошарова нейронна мережа

Навчання сучасних багатошарових нейронних мереж здійснюється методом *зворотного розповсюдження помилки* (*backpropagation*). Коротко суть даного алгоритма полягає в тому, що мережі на вхід подається деякий образ(и) і оцінюється її реакція – значення на виході. Далі за допомогою спеціальної функції – *функції втрат* визначається *помилка мережі* – оцінка того, наскільки мережа дає некоректний відгук на даних, що були подані на її вхід. На основі знайденої величини помилки, починаючи з вихідного шару розраховується *градієнт помилки* відносно параметрів кожного нейрона. Для оцінки градієнтів параметрів у проміжних (прихованих) шарах використовується *ланцюгове правило диференціювання*, при цьому також оцінюються градієнти значень виходів нейронів у проміжних шарах. Знайдені градієнти можуть усереднюватись по певній невеликій підвибірці (minibatch) навчальних образів. Далі, на основі знайдених градієнтів, користуючись методом *стохастичного градієнтного спуску* (Stochastic Gradient Descent – SGD) виконується переналаштування параметрів нейронів, так щоб мінімізувати помилку мережі.

Серед функцій втрат (помилки), що використовуються при розпізнаванні образів, найбільшого розповсюджених набули: сума квадратів різниць, функція втрат SVM (hinge loss) та взаємна-ентропія. В загальному випадку функція втрат може бути будь-якою і залежить від конкретної задачі, що вирішується.

Найбільш розповсюдженими модифікаціями методу стохастичного градієнтного спуску (SGD), що застосовуються при навчанні сучасних нейронних мереж є: стохастичний градієнтний спуск з імпульсом (momentum), метод Нестерова, AdaGrad, RMSProp та Adam.

На сьогодні повнозв’язні нейронні мережі, які мають структуру подібну до тої, що показана на рис. 7.3 в чистому вигляді майже не використовуються. Натомість в сучасних задачах комп’ютерного зору застосовуються *глибокі* нейронні мережі зі згортковими шарами. Глибокими

умовно вважаються мережі, які мають щонайменше 5-7 шарів. На рис. 7.4 показано один з найпростіших варіантів мереж даного типу.

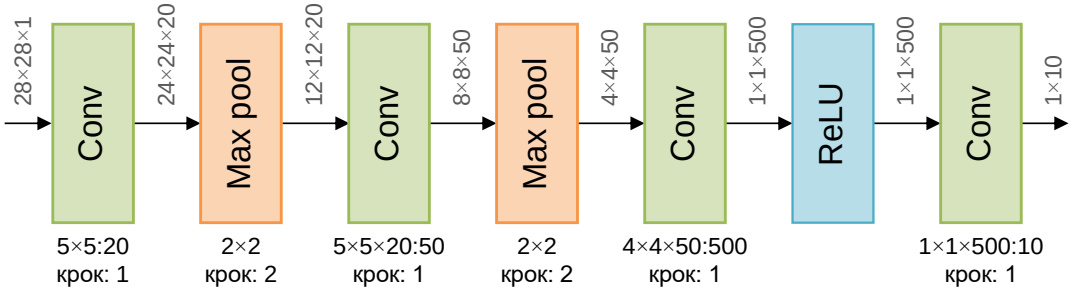

Рис. 7.4 – Проста LeNet-подібна згорткова нейронна мережа

У наведеній вище штучній нейронній мережі використані наступні позначення: *Conv* – згорткові шари, *Max pool* – шари об'єднання за максимумом, *ReLU* – функція активації $a = \max(n, 0)$. Позначення під кожним із шарів мають формат: $x \times y \times z : n$, $x \times y : n$ та $x \times y$ і означають наступне: $x \times y \times z$ та $x \times y$ – розміри фільтру в шарі; n – кількість фільтрів у шарі; *крок* – зміщення фільтрів у пікселях під час обчислення згортки або при об'єднанні за максимумом. Над стрілочками, що сполучають шари, стоять розмірності масивів (тензорів) даних, які передаються між відповідними шарами.

Навчання та тестування згорткових нейронних мереж в середовищі MATLAB можна виконувати за допомогою зовнішнього безкоштовного пакету, що має назву MATCONVNET. Даний пакет необхідно скачати, а потім скомпілювати згідно інструкцій наданих на офіційному сайті MATCONVNET (http://www.vlfeat.org/matconvnet/). Для роботи із сучасними нейронними мережами у Python існує чимало готових пакетів, серед яких можна обрати будь-який – за власним бажанням. Тим не менш рекомендованими є пакети PyTorch (https://pytorch.org) або TensorFlow (https://www.tensorflow.org).

Створення звичайних повнозв'язних багатошарових мереж (багатошарових персептронів) у системі MATLAB можна здійснювати функцією **`feedforwardnet`** (в старих версіях **`newff`**), їх навчання виконувати функцією **`train`**, а застосування вже навченої мережі – функцією **`sim`**.

Зауважимо, що для надійного навчання нейронних мереж їх навчальні вибірки мають містити в залежності від задачі від *декількох десятків тисяч* до *декількох десятків мільйонів* різних прикладів.

Порядок виконання роботи

1. Згідно наведених нижче завдань здійснити розпізнавання зображень.
2. Дослідити як параметри методів розпізнавання впливають на отримуваний результат.
3. Представити процедури розпізнавання зображень у вигляді файла-скрипта.

Завдання

Виконати обробку заданих зображень:

1. Вибрати неоднорідну область на вихідному зображенні та зберегти її. Спотворити вихідне зображення додавши шуми, незначне обертання, афінні (або проективні) деформації, тощо. Здійснити пошук збереженої області на спотвореному зображенні за допомогою кореляційного співставлення (файл – cameraman.tif, що входить до складу MATLAB).
2. В обраному пакеті для навчання нейронних мереж (MATCONVNET, PyTorch, Tensorflow чи іншому) створити штучну згорткову нейронну мережу (подібну до тої, що показана на рис. 7.4) для розпізнавання рукописних цифр на навчальній базі MNIST[1] (всі перелічені пакети для навчання нейронних мереж мають у своєму складі відповідний приклад). Протестувати навчену мережу на прикладах.
3. За бажанням дослідити інші наявні в MATLAB методи машинного навчання для розпізнавання зображень.

---

[1] Базу MNIST можна завантажити за посиланням: http://yann.lecun.com/exdb/mnist/

Запитання для самоконтролю

1. Наведіть означення понять «образ», «ознака» та «клас». В якому вигляді зазвичай представляють ознаки.
2. У чому полягає суть розпізнавання основаного на співставленні. Назвіть метрики (міри) відстані, що набули найбільшого застосування при обробці зображень.
3. Як здійснюється класифікація за мінімумом відстані. Що є прототипом образу в даному виді класифікатора.
4. В чому полягає перевага коефіцієнту кореляції в порівнянні зі звичайною кореляцією.
5. Одношаровий персептрон – в чому суть процедури навчання. Який основний недолік одношарового персептрону.
6. Наведіть найбільш поширені функції активації штучних нейронів.
7. Наведіть рівняння в матричному вигляді, за яким обчислюється відгук багатошарової нейронної мережі (для спрощення достатньо буде навести рівняння для двошарової або тришарової мережі).
8. За якою процедурою здійснюється навчання багатошарового персептрона.
9. Наведіть найбільш розповсюджені види нейронних мереж.
10. Опишіть інші відомі Вам методи машинного навчання.

ДОДАТОК А

Робота в системі MATLAB та GNU Octave

Що таке Matlab?

MATLAB – це середовище та високопродуктивна інтерпретована мова для технічних розрахунків. MATLAB дозволяє виконувати обчислення, візуалізацію і програмування в зручному вигляді, що є найбільш близьким до математичного.

Типове використання MATLAB:

- ✓ математичні обчислення;
- ✓ створення алгоритмів;
- ✓ моделювання;
- ✓ аналіз даних, дослідження та візуалізація;
- ✓ наукова та інженерна графіка;
- ✓ розробка додатків, включаючи створення графічного інтерфейсу.

MATLAB – це інтерактивна система, в якій основним елементом даних є масив. Це дозволяє вирішувати різні завдання, пов'язані з технічними обчисленнями, особливо в яких використовуються матриці та вектора, у декілька разів швидше, ніж при написанні програм з використанням «скалярних» мов програмування, таких як **C** або **Фортран**.

Слово MATLAB означає матрична лабораторія (**MATrix LABoratory**). MATLAB був спеціально написаний для забезпечення легкого доступу до **LINPACK** і **EISPACK**, які є сучасними програмними засобами для матричних обчислень.

MATLAB розвивався протягом декількох десятиріч, орієнтуючись на різних користувачів. В академічному середовищі, він був стандартним інструментом для роботи у різних областях математики, машинобудування, хімії, біології, а також інших наук. У промисловості, MATLAB – це інструмент для високопродуктивних досліджень, розробки алгоримів і аналізу даних.

У MATLAB важлива роль відводиться спеціалізованим групам програм, які називаються *тулбоксами* (*toolboxes*). *Тулбокси* – це бібліотеки функцій, які дозволяють вирішувати специфічні класи завдань. Зокрема, вони застосовуються для вирішення задач обробки сигналів, систем контролю, нейронних мереж, нечіткої логіки, вейвлетів, моделювання тощо.

Складові системи Matlab

Система MATLAB складається з п’яти основних елементів:

1. Мова MATLAB. Це мова матриць та масивів високого рівня з управлінням потоками, функціями, структурами даних, введенням-виводом і особливостями об’єктно-орієнтованого програмування. Це дозволяє як програмувати у «невеликому масштабі» для швидкого створення чорнових програм, так і в «великому» для створення великих і складних додатків.
2. Середовище MATLAB. Це набір інструментів і додатків, з якими працює користувач або програміст MATLAB. Воно включає засоби для управління змінними в робочому просторі MATLAB, введенням і виведенням даних, а також створення, контролю і відладки M-файлів і додатків MATLAB.
3. Керована графіка. Це графічна система MATLAB, яка включає команди високого рівня для візуалізації дво- і тривимірних даних, обробки зображень, анімації і ілюстрованої графіки. Вона також включає команди низького рівня, що дозволяють повністю редагувати зовнішній вигляд графіки, також як при створенні Графічного Інтерфейсу Користувача (GUI) для MATLAB додатків.
4. Бібліотека математичних функцій. Це широка колекція обчислювальних алгоритмів від елементарних функцій, таких як сума, синус, косинус, комплексна арифметика, до складніших, зокрема, обернення матриць, знаходження власних значень, функції Бесселя, швидке перетворення Фур’є тощо.

5. Програмний інтерфейс. Це бібліотека, яка дозволяє писати програми на C та Фортрані, які взаємодіють з MATLAB. Вона включає засоби для виклику програм з MATLAB (динамічний зв'язок), викликаючи MATLAB як обчислювальний інструмент і для читання-запису MAT-файлів.

Система Simulink

SIMULINK, супутня програма MATLAB, що є інтерактивною системою для моделювання нелінійних динамічних систем. SIMULINK – середовище, яке дозволяє моделювати процеси шляхом створення діаграм і їх маніпуляцією (фактично це графічна мова програмування подібна до мови G – LabView). SIMULINK працює з лінійними, нелінійними, безперервними, дискретними, багатовимірними системами.

Blocksets – це бібліотеки блоків, що розширюють можливості SIMULINK для при вирішення спеціалізованих задач, таких як зв'язок, обробка сигналів, енергетичні системи тощо (фактично це аналог тулбоксів MATLAB, причому інколи блоксети постачаються разом із відповідними тулбоксами, надаючи SIMULINK такої ж функціональності, що й MATLAB).

Система GNU Octave

GNU Octave є системою комп'ютерних розрахунків є аналогом MATLAB із відкритим вихідним кодом. Інтерпретатор системи GNU Octave сумісний із синтаксисом мови MATLAB (проте має деякі додаткові розширення). Функціональність GNU Octave може бути розширена за рахунок *пакетів* (аналог тулбоксів MATLAB). Оскільки MATLAB та GNU Octave є сумісними, більшість тулбоксів працюватимуть на GNU Octave. Водночас пакети Octave через використання розширених його можливостей на MATLAB найімовірніше працювати не будуть.

Наразі в GNU Octave відсутнє середовище для моделювання та графічного програмування аналогічне до SIMULINK. Тим не менш, розвиток GNU Octave триває.

Матриці

Основним типом змінних в MATLAB є *матриця* – прямокутний масив чисел (зазвичай з плаваючою точкою). Для представлення *скалярних значень* тут використовуються матриці розміром 1×1, а *векторів* – матриці, що мають один єдиний стовпчик або рядок. MATLAB використовує різні способи для зберігання чисельних і не чисельних даних (текстові рядки, структури, комірки, таблиці), однак для спрощення всі дані краще вважати представленими у матричному вигляді, оскільки мова MATLAB організована так, щоб всі операції в ній були якомога природнішими: в той час як більшість мов програмування працюють з числами як елементами мови, MATLAB дозволяє дуже швидко і ефективно оперувати одразу матрицями.

Введення матриць

Введення матриць в MATLAB може виконуватись декількома способами:

- вводити повний список елементів;
- завантажувати матриці із зовнішніх файлів;
- генерувати матриці, використовуючи вбудовані функції;
- створювати матриці за допомогою власних функцій в M-файлах.

Починаємо з введення матриці як списку елементів. Для цього слід дотримуватись ряду правил, а саме:

- відокремлювати елементи рядка пробілами або комами «`,`»;
- використовувати крапку з комою «`;`» для позначення закінчення кожного рядка;
- слідкувати, щоб кожен рядок містив однакову кількість елементів;
- оточувати весь список елементів квадратними дужками `[]`.

Щоб ввести матрицю достатньо записати в командному вікні середовища наступний код (рис. 1.1):

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB відобразить матрицю, яку було введено:

```
A =

    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

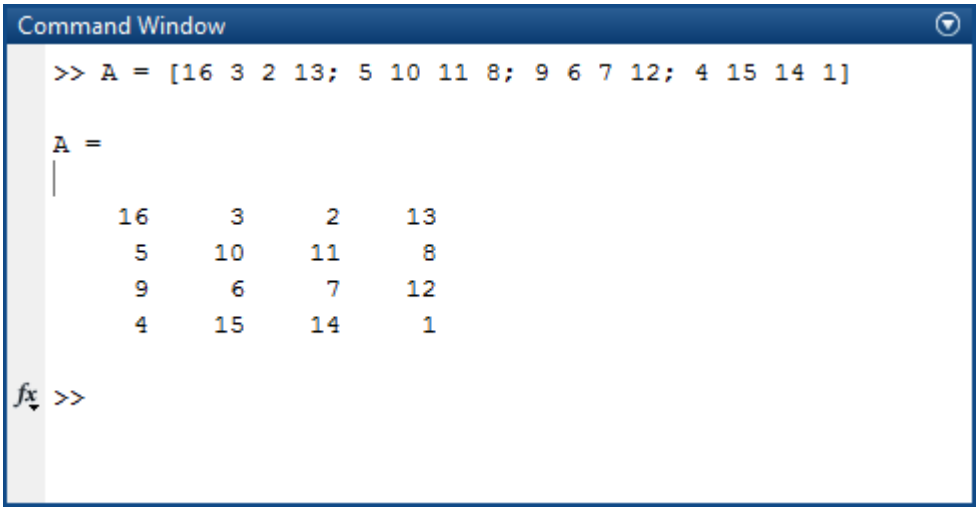

Рисунок А.1 – Приклад фрагмента командного вікна MATLAB

Після введення матриці, вона автоматично запам'ятовується середовищем MATLAB, і до неї можна легко звертатись за ім'ям **A**. Зараз ми маємо **A** в робочому просторі MATLAB (рис. 1.2).

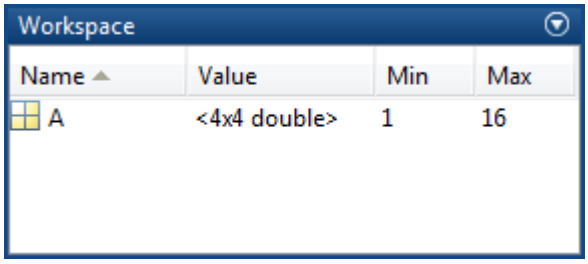

Рисунок А.2 – Приклад фрагмента робочого простору MATLAB

Операції підсумовування елементів, транспонування і діагоналізації матриці

Введена вище матриця називається «магічним квадратом» і володіє рядом цікавих властивостей пов'язаних з різними способами підсумовування його елементів. Зокрема, якщо взяти суму елементів уздовж якого-небудь рядка, стовпця, або уздовж будь-якої з головних його

діагоналей, завжди буде виходити одне і те саме число. Перевіримо це користуючись MATLAB. Для цього скористаємось функцією пошуку суми:

```
sum(A)
```

Система MATLAB видасть наступну відповідь:

```
ans =

    34    34    34    34
```

Коли вихідна змінна не визначена, MATLAB використовує змінну **ans**, коротко від «*answer*» – відповідь, для зберігання результатів обчислення. Ми підрахували *вектор-рядок*, що містить суму елементів *стовпців* матриці **A**. Дійсно, кожен стовпець має однакову суму, магічну суму, рівну 34.

А як щодо сум в рядках? MATLAB за замовчуванням надає перевагу роботі зі стовпчиками матриці. Таким чином, очевидний спосіб отримати суму в рядках – це транспонувати матрицю, підрахувати суму в стовпцях, а потім транспонувати результат. Операція транспонування позначається апострофом або одинарною лапкою. Вона дзеркально відображує матрицю відносно головної діагоналі, змінюючи рядки на стовпці:

```
A'

ans =

    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

А вираз

```
sum(A')'
```

викликає результат вектор-стовпець, що містить суми в рядках:

```
ans =

    34
    34
    34
    34
```

Крім того, суму елементів рядків також можна знайти одразу за допомогою функції **sum**, передаючи їй додатковий аргумент, що вказує по якій розмірності слід шукати суму:

```
sum(A, 2)
```

дасть такий самий результат.

Суму елементів на головній діагоналі можна легко отримати за допомогою функції **diag**, яка вибирає цю діагональ.

```
diag(A)

ans =

    16
    10
     7
     1
```

А функція:

```
sum(diag(A))
```

дає:

```
ans =

    34
```

Таким чином, ми перевірили, що матриця на гравюрі Дюрера дійсно магічна, і навчилися використовувати деякі матричні операції MATLAB. У подальших розділах ми продовжимо використовувати цю матрицю для демонстрації додаткових можливостей MATLAB.

Індекси

Елемент в рядку *i* і стовпчику *j* матриці **A** позначається **A(i, j)**. Наприклад, **A(4, 2)** – це число в четвертому рядку і другому стовпці. Для нашого магічного квадрата **A(4, 2) = 15**. Таким чином, можна обчислити суму елементів в четвертому стовпці матриці **A**, набравши:

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

```
ans =

    34
```

Проте це не найкращий спосіб підсумовування окремого рядка. Також можна звертатися до елементів матриці через один індекс, **A(k)**. Це звичайний спосіб посилання на рядки і стовпці матриці. Але його можна використовувати тільки з двовимірними матрицями. В цьому випадку масив розглядується як довгий вектор, сформований із стовпців початкової матриці.

Так, для нашого магічного квадрата, **A(8)** – це інший спосіб посилатися на значення 15, що зберігається в **A(4, 2)**. Якщо ви намагаєтеся використовувати значення елементу поза матрицею, MATLAB видасть помилку:

```
t = A(4,5)

??? Index exceeds matrix dimensions.
```

З іншого боку, якщо зберігати значення за межами матриці, то її розмір збільшується.

```
X = A;

X(4, 5) = 17

X =

    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

Оператор двокрапки

Двокрапка «:» – це один з найбільш важливих операторів MATLAB. Він проявляється в різних формах. Вираз:

```
1:10

ans =

     1     2     3     4     5     6     7     8     9    10
```

Для отримання зворотного інтервалу, опишемо приріст. Наприклад

```
100:-7:50
ans =

   100    93    86    79    72    65    58    51
```

або

```
0:pi/4:pi
```

що дає:

```
ans =
     0    0.7854    1.5708    2.3562    3.1416
```

Індексний вираз, включаючи двокрапку, відноситься до частки матриці. **A(1:k, j)** – це перші до елементів *j*-го стовпця матриці **A**.

Так **sum(A(4, 1:4))** обчислює суму четвертого рядка. Але є і кращий спосіб. Двокрапка, сама по собі, звертається до всіх елементів в рядку і стовпці матриці, а слово **end** – до останнього рядка або стовпця.

```
sum(A(:,end))
```

обчислює суму елементів в останньому стовпці матриці **A**

```
ans =
    34
```

Чому магічна сума квадрата 4×4 рівна 34? Якщо цілі числа від 1 до 16 відсортовані в чотири групи з рівними сумами, ця сума має бути

```
sum(1:16) / 4
```

яка, звичайно, рівна:

```
ans =

    34
```

Якщо оператор двокрапки просто передати як індекс масиву, то результатом такої операції буде вектор-стовпчик, який складається зі стовпчиків вихідного масиву, вишикуваних один за одним по порядку:

```
a = [1 2 3; 4 5 6; 7 8 9]

a =
```

```
     1     2     3
     4     5     6
     7     8     9

a(:)

ans =

     1
     4
     7
     2
     5
     8
     3
     6
     9
```

Вирази

Як і більшість інших мов програмування, MATLAB надає можливість виконання математичних операцій, але як вже відзначалося ці операції можуть безпосередньо виконуватись над матрицями, а також над багатомірними масивами. Основні складові виразів мови MATLAB це:

- змінні;
- числа;
- оператори;
- функції.

Змінні

В MATLAB немає необхідності у визначенні типу змінних або розмірності. Коли MATLAB зустрічає нове ім’я змінної, він автоматично створює змінну і виділяє відповідний об’єм пам’яті. Якщо змінна вже існує, MATLAB змінює її і якщо це необхідно виділяє додаткову пам’ять. Наприклад:

```
num_students = 25
```

створює матрицю 1×1 з ім’ям **num_students** і зберігає значення 25 в її єдиному елементі.

Імена змінних складаються з букв, цифр або символів підкреслення. MATLAB використовує тільки перші 31 символ імені змінної. MATLAB **чутливий до регістрів**, він розрізняє заголовні і рядкові букви. Тому «**A**» і «**a**» – не одна і та ж змінна. Щоб побачити матрицю пов'язану із змінною, необхідно просто ввести відповідну назву змінної.

Числа

MATLAB використовує звичайну десяткову систему числення, з необов'язковою десятковою крапкою і знаками плюс-мінус для чисел. Наукова система числення використовує букву **e** для визначення показника ступені десяти. Уявні числа використовують **i** або **j** як суфікс.

Всі числа (якщо явно не задано) по замовчуванню використовують для зберігання формат **double**, визначений стандартом IEEE 764. Числа з плаваючою крапкою володіють обмеженою точністю – приблизно 16 значущих цифр і обмеженим діапазоном – приблизно від 10^{-308} до 10^{+308}.

Оператори

У виразах MATLAB використовуються звичайні арифметичні оператори зі звичайним порядком дій.

+	додавання
–	віднімання
*****	множення
/	ділення
****	ліве ділення (див. документацію)
^	степінь
'	транспонування
()	зміна порядку дій, доступ до елементів масиву, виклик функцій

Підтримуються також логічні оператори, а також оператори по-елементних дій, які визначені для деяких простих операторів і мають синтаксис «**.***», де ***** – це простий оператор.

Функції

MATLAB надає велику кількість елементарних математичних функцій, таких як **abs**, **sqrt**, **exp**, **sin**. Обчислення квадратного кореня або логарифма негативного числа не є помилкою: в цьому випадку результатом є відповідне комплексне число. MATLAB також надає і складніші функції, включаючи Гамма-функцію і функції Бесселя. Більшість з цих функцій мають комплексні аргументи. Щоб вивести список всіх елементарних математичних функцій, наберіть:

```
help elfun
```

Для виведення складніших математичних і матричних функцій, наберіть:

```
help specfun
```

```
help elmat
```

відповідно.

Деякі функції, такі як **sqrt** і **sin**, – вбудовані. Вони є частиною MATLAB, тому вони дуже ефективні, але їх обчислювальні деталі важко доступні. Тоді як інші функції, такі як **gamma** і **sinc**, реалізовані в М-файлах. Тому ви можете легко побачити їх код і, у разі потреби, навіть модифікувати його.

Константи

pi	3,14159265…
i	уявна одиниця
j	те ж саме, що й *i*
eps	відносна точність числа з плаваючою крапкою
realmin	найменше число з плаваючою крапкою
realmax	найбільше число з плаваючою крапкою
Inf	нескінченність
NaN	не число (виникає, наприклад, при діленні 0 / 0)

Нескінченність з’являється при діленні на нуль або при виконанні математичного виразу, що приводить до переповнювання, тобто до

перевищення **realmax**. Не число (**NaN**) генерується при обчисленні виразів типу **0/0** або **Inf - Inf**, які не мають певного математичного значення.

Імена функцій не є зарезервованими, тому можливо змінювати їх значення на нові, наприклад:

```
eps = 1.e-6
```

і далі використовувати це значення в подальших обчисленнях. Початкове значення може бути відновлене таким чином

```
clear eps
```

Вирази

Ви вже ознайомились із деякими прикладами використання виразів MATLAB. Нижче наведено ще декілька прикладів з результатами.

```
rho = (1 + sqrt(5)) / 2
rho =

1.6180


a = abs(3 + 4i)
a =
5

z = sqrt(besselk(4 / 3, rho - i))
z =
0.3730 + 0.3214i

huge = exp(log(realmax))
huge = 1.7977e+308

toobig = pi * huge
toobig = Inf
```

Створення матриць

MATLAB має наступні основні функції, які створюють матриці:

`[]`	пуста матриця
`zeros`	матриця, всі елементи якої нулі
`ones`	матриця, всі елементи якої одиниці
`eye`	одинична матриця (містить одиниці на головній діагоналі)
`rand`	рівномірний розподіл випадкових елементів
`randn`	нормальний розподіл випадкових елементів
`false`	матриця логічних нулів
`true`	матриця логічних одиниць

Якщо функції, що створює матрицю буде передано один аргумент, то буде згенеровно квадратну матрицю з кількістю стовпчиків і рядків рівній значенню аргументу.

Наведемо деякі приклади:

```
Z = zeros(2,4)
Z =

     0     0     0     0
     0     0     0     0

F = 5*ones(3,3)
F =
     5     5     5
     5     5     5
     5     5     5

N = round(10 * rand(1,10))
N =
     8     9     1     9     6     1     3     5    10    10

R = randn(4,4)
R =
   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139

E = eye(3)
E =
     1     0     0
     0     1     0
     0     0     1
```

Завантаження матриць

Команда **load** зчитує двійкові файли, що містять матриці, створені в MATLAB раніше, або текстові файли, що містять чисельні дані. Текстові файли мають бути сформовані у вигляді прямокутної таблиці чисел, відокремлених пропусками, з рівною кількістю елементів в кожному рядку. Наприклад, створимо зовні MATLAB текстовою файл, що містить 4 рядки:

```
16.0     3.0     2.0    13.0
 5.0    10.0    11.0     8.0
 9.0     6.0     7.0    12.0
 4.0    15.0    14.0     1.0
```

Збережемо цей файл під ім'ям **magik.dat**. Тоді команда **load magik.dat** прочитає цей файл і створить змінну **magik**.

Об'єднання матриць та масивів

Об'єднання – це процес з'єднання маленьких матриць для створення великих. Фактично, ви створили вашу першу матрицю об'єднанням її окремих елементів. Пара квадратних дужок – це оператор об'єднання. Наприклад, почнемо з матриці **A** (магічного квадрата 4×4) і сформуємо:

```
B = [A, A + 32; A + 48, A + 16]
```

Результатом буде матриця 8×8, що отримується з'єднанням чотирьох підматриць:

```
B =

    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

Це матриця лише наполовину є магічною. Її елементи є комбінацією цілих чисел від 1 до 64, а суми в стовпцях точно дорівнюють значенню для магічного квадрата 8×8.

```
sum (B)

ans =

   260   260   260   260   260   260   260   260
```

Проте, суми в рядках цієї матриці (`sum(B')'`) не всі однаково. Необхідно провести додаткові операції, щоб зробити цю матрицю дійсно магічним квадратом 8×8.

Звернення до елементів масивів

Для звернення до елементів масивів у MATLAB використовується оператор круглих дужок. В дужках через кому наводяться положення елементу вздовж кожної розмірності матриці.

Наприклад:

```
A = rand(1, 3, 4)

A(:,:,1) =

    0.1190    0.4984    0.9597

A(:,:,2) =

    0.3404    0.5853    0.2238
A(:,:,3) =

    0.7513    0.2551    0.5060

A(:,:,4) =

    0.6991    0.8909    0.9593

A(1, 2, 4)

ans =

    0.8909
```

Якщо необхідно отримати не один єдиний елемент, а частину масиву (підмасив), то в дужках можна наводити масиви, що містять положення необхідних елементів вздовж кожної розмірності. Для автоматичного генерування цих масивів доцільно користуватись оператором двокрапки «**:**». Наведемо приклад:

```
A = magic(5)

A =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

B = A(2:4, 2:5)

B =

     5     7    14    16
     6    13    20    22
    12    19    21     3

B([1, 3], [2, 4])

ans =

     7    16
    19     3
```

Для спрощення індексації можуть бути використані спеціальні позначення: лише один оператор двокрапки «**:**» означає вибір всіх елементів вздовж відповідної розмірності, а оператор «**end**» – останній елемент. При цьому запис **A(end, :)** означає вибір всього останнього рядка матриці **A**, запис **A(:, 3:end)** – у всіх рядках масиву обрати з 3 по останній стовпчики. Наприклад:

```
A = rand(3, 4)

A =

    0.5472    0.2575    0.8143    0.3500
    0.1386    0.8407    0.2435    0.1966
    0.1493    0.2543    0.9293    0.2511
```

```
A(end, :)

ans =

    0.1493    0.2543    0.9293    0.2511

A(:, 3:end)

ans =

    0.8143    0.3500
    0.2435    0.1966
    0.9293    0.2511
```

При цьому, очевидно, результати виконання операцій індексації `A(3, :)` та `A(3, 1:end)` будуть однаковими.

Для звернення до елементів масиву будь-якої розмірності також можна використовувати порядкові номери даних елементів. Для цього при індексації в дужках наводиться один єдиний аргумент – масив номерів необхідних елементів. Наприклад:

```
A = magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

A([5, 10, 11])

ans =

     2    10     6
```

Розрахунок положення в даному випадку ведеться по стовпчикам масиву. Тобто в наведеному вище прикладі першим за порядком елементом є число 16, другим – 5, третім – 9, четвертим – 4, п’ятим – 2 і т.д.

Наведемо ще один приклад для багатомірного (3-мірного) масиву:

```
A = rand(2, 3, 3)

A(:,:,1) =

    0.5688    0.0119    0.1622
```

```
    0.4694    0.3371    0.7943

A(:,:,2) =

    0.3112    0.1656    0.2630
    0.5285    0.6020    0.6541

A(:,:,3) =

    0.6892    0.4505    0.2290
    0.7482    0.0838    0.9133

A(10)

ans =

    0.6020
```

Якщо при індексації у дужках масиву будь-якої розмірності записати єдиний аргумент двокрапку, то такий масив буде перетворено на вектор стовпчик, причому це буде зроблено по стовпцях, далі по строках і далі за порядком по всім іншим розмірностям масиву. Наприклад:

```
A = rand(3, 3, 2)

A(:,:,1) =

    0.1524    0.9961    0.1067
    0.8258    0.0782    0.9619
    0.5383    0.4427    0.0046


A(:,:,2) =

    0.7749    0.0844    0.8001
    0.8173    0.3998    0.4314
    0.8687    0.2599    0.9106

A(:)

ans =

    0.1524
    0.8258
    0.5383
    0.9961
    0.0782
    0.4427
    0.1067
    0.9619
```

```
    0.0046
    0.7749
    0.8173
    0.8687
    0.0844
    0.3998
    0.2599
    0.8001
    0.4314
    0.9106
```

Ще одним способом звернутися до елементів масиву є використання двійкового масиву такої ж розмірності. Двійковий масив записується у дужках вихідного масиву, при цьому він повинен містити *одиниці* в тих елементах, до яких необхідно отримати доступ. Наведемо приклад:

```
A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

B = logical([0 1 0; 1 0 0; 0 0 1])

B =

     0     1     0
     1     0     0
     0     0     1
A(B)

ans =

     4
     2
     9
```

Видалення рядків і стовпців

Видаляти рядки і стовпчики матриці можна, використовуючи просто пару квадратних дужок.

```
X = magic(4);
```

Тепер видалимо другий стовпчик матриці **X**.

```
X(:, 2) = []
```

Ця операція змінить **X** таким чином:

```
X =

    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

Якщо необхідно видалити один елемент матриці, то результат вже не буде матрицею. Тому вираз:

```
X(1, 2) = []
```

видасть помилку. Проте використання одного індексу видаляє окремий елемент або послідовність елементів і перетворить елементи, що залишилися, у вектор-рядок:

```
X(2:2:10) = []
X =

    16     9     2     7    13    12     1
```

Арифметичні дії на матрицями на прикладі множення

Перемноження матриць за правилами лінійної алгебри виконується за допомогою оператора «*». Зокрема:

```
A =

    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
B =

    16     4     7     3
     5    -7     2     9
     0     8    23    65
    -7     4    17     9
```

Тоді **C = A * B** дасть результат:

```
C =

         180         111         385         322
          74          70         444         892
          90          98         440         644
         132          27         397        1066
```

Також в системі MATLAB передбачена можливість по-елементного перемножування. Для цієї мети використовується крапка перед знаком множення. Наприклад:

```
C = A .* B
```

в результаті отримаємо:

```
C =

   256    12    14    39
    25   -70    22    72
     0    48   161   780
   -28    60   238     9
```

Деякі корисні при роботі з матрицями функції

repmat(...)	утворює матрицю, яка складається із повторень матриці-аргумента заданої кількості раз у заданих розмірностях.
reshape(...)	змінює розміри матриці.
meshgrid(...)	генерує двомірну сітку.
rot90(...)	розвертає матрицю проти годинникової стрілки на 90 градусів
flipud(...)	перевертає двомірну матрицю зверху вниз так, що перший рядок стає останнім, а останній – першим.
fliplr(...)	перевертає двомірну матрицю зліва на право так, що лівий стовпчик стає правим, а правий – лівим.
flipdim(...)	перевертає матрицю вздовж заданої розмірності.
permute(...)	змінює порядок розмірностей у багатовимірному масиві.
circshift(...)	циклічний зсув елементів у матриці.

MATLAB має ще величезну кількість функцій для зручної обробки матриць. З ними можна детально ознайомитись у допомозі системи MATLAB.

Для того щоб дізнатися яку розмірність та розміри має багатомірний масив чи матриця корисними можуть бути наступні функції:

size(...)	повертає вектор кількостей (кількість) елементів вздовж кожної (заданої) розмірності.
numel(...)	загальна кількість елементів у масиві (по всім розмірностям разом).
ndims(...)	кількість розмірностей у масиві, що є аргументом даної функції.

Наприклад:

```
a = ones(2, 3, 5);

ndims(a)
ans =
     3

size(a)
ans =
     2     3     5

size(a, 3)
ans =
     5

numel(a)
ans =
    30
```

Створення m-файлів

М-файли є звичайними текстовими файлами, які створюються за допомогою текстового редактора. Система MATLAB має спеціальний вбудований редактор/відладник, проте за бажанням можна використовувати і будь-який інший текстовий редактор.

Відкрити редактор можна двома способами:

- Кнопкою New Script в нових версіях, з меню File вибравши опцію New, а потім M-File – в старих версіях MATLAB або натиснувши Ctrl + N.
- Використовувати команду редагування **edit**.

М-функції є m-файлами, які допускають наявність вхідних і вихідних аргументів. Вони працюють із змінними в межах власної робочої області, відмінної від робочої області системи MATLAB.

Приклад: Функція **average** – це простий М-файл, який обчислює середнє значення елементів вектора:

```
function y = average(x)
    % AVERAGE Середнє значення елементів вектора.
    % AVERAGE(X), де X - вектор. Обчислює середнє значення
    % елементів вектора.
    % Якщо вхідний аргумент не є вектором, генерується помилка.

    [m, n] = size(x);
    if (~((m == 1) || (n == 1)))
        error('Вхідний масив має бути вектором');
    end

    y = sum(x) / length(x);     % Власне обчислення
```

Спробуємо ввести ці команди в М-файл, названий **average.m**. Функція **average** допускає єдиний вхідний і єдиний вихідний аргументи. Для того, щоб викликати функцію **average**, треба ввести наступні команди:

```
z = 1:99;
average(z)
```

Отримуємо результат:

```
ans = 50
```

Операції вводу-виводу зображень

`imread('filename')`	завантаження зображення.
`imwrite(f, 'filename')`	збереження зображення.
`imshow(f)`	відображення зображення.
`iminfo 'filename'`	отримання інформації про графічний файл.

Приклад:

```
img = imread('cameraman.tif');
imwrite(img, 'myimage.png');
imshow(img);
```

Після виконання функції **imshow** на екрані має з’явитись зображення показане на рис. А.3.

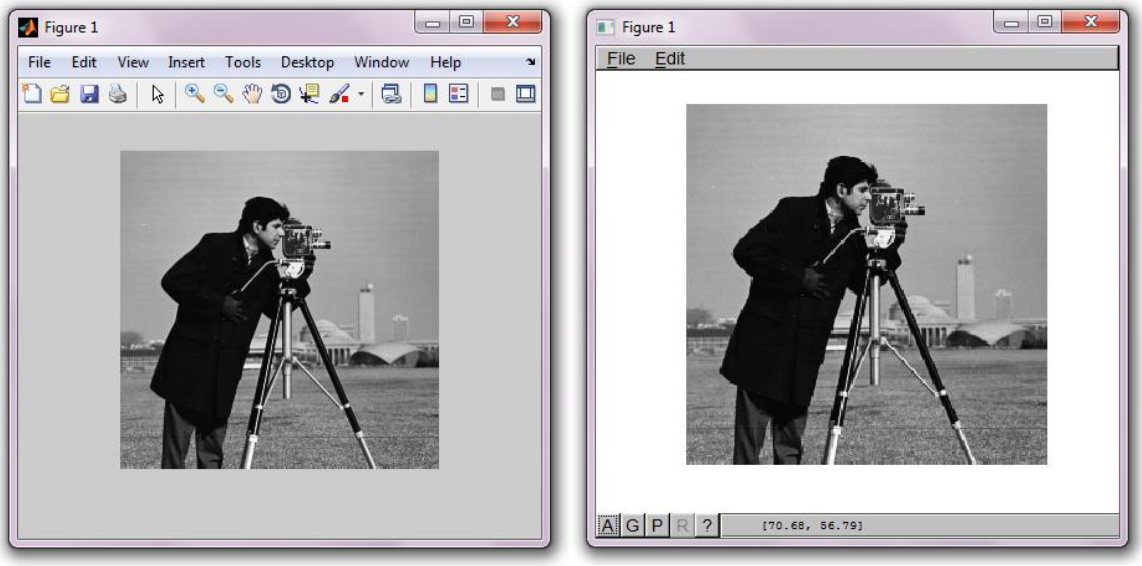

Рисунок А.3 – Виведене функцією **`imshow`** зображення
зліва у MATLAB справа у GNU Octave

Рекомендації щодо оформлення лабораторних робіт

Лабораторні роботи починаючи з другої міститимуть декілька різних завдань. З метою їх зручного їх демонстрації лабораторну роботу доцільно представляти скриптом (m-файлом). При цьому якщо використовується система MATLAB можливо два варіанти:

1. Кожне завдання оформлюється як окремий m-файлом та запускається через командну строку або з редактора файлів;
2. Всі завдання оформлюються в одному файлі та розділюються коментарем вигляду «`%%`» – роздільником секцій. Задаючи курсором секцію (активна секція підсвічується іншим кольором) та використовуючи команди редактора Run Section і Run and Advance (див. рис. А.4), кожну секцію можна виконувати окремо.

При роботі в MATLAB бажано використовувати другий варіант оформлення – з роздільниками секцій.

Рисунок А.4 – Команди виконання секцій у MATLAB
в старому редакторі – зліва, в новому – справа

Приклад m-файлу, що використовує роздільники секцій наведено нижче.

```
%% Читання файлу зображення
file = 'cameraman.tif';
i = imread(file);

%% Вивід зображення на екран
imshow(i);
```

Завдання та задачі для самоконтролю

1. Згенерувати матрицю заданої розмірності без використання циклів, що має вигляд:

$$\left[\begin{array}{ccc:ccc} 1 & \cdots & 1 & 2 & \cdots & 2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & 1 & 2 & \cdots & 2 \\ \hdashline 3 & \cdots & 3 & 4 & \cdots & 4 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 3 & \cdots & 3 & 4 & \cdots & 4 \end{array}\right]$$

2. Замінити значення головної діагоналі матриці на задані значення не користуючись операторами циклів.
3. Видалити задані рядки чи/або стовпчики матриці не користуючись операторами циклів.
4. Зберегти, завантажити та відобразити на екрані зображення.

ДОДАТОК Б

Робота на мові Python

Пакети необхідні для роботи з мовою Python

Для роботи на мові Python необхідні наступні додаткові пакети:

- ✓ numpy
- ✓ scipy
- ✓ scikit-image
- ✓ matplotlib

Ті хто хоче максимально швидко розпочати роботу з Python, може скачати готовий дистрибутив, що містить всі необхідні пакети за посиланням:

https://www.anaconda.com/download/

Наведена далі інформація буде корисна тим, хто вже має Python, але його дистрибутив не містить необхідних пакетів, а також тим, хто з якоїсь причини хоче встановити Python та пакети вручну.

Встановлення пакетів виконується за допомогою скрипта **pip** з командного рядка, наприклад:

```
pip install numpy
python -m pip install {ШляхДоЛокальногоПакету.whl}
```

В першому випадку пакет буде скачуватись з репозиторію та встановлюватись автоматично.

При роботі на операційній системі Windows відповідні пакети можна завантажити за посиланням:

http://www.lfd.uci.edu/~gohlke/pythonlibs/

Якщо виявилося, що менеджер пакетів **pip** не встановлено, скористайтеся документацією з його встановлення за наступним посиланням:

https://pip.pypa.io/en/stable/installing/

Імпорт пакетів та доступ до функцій

У Python на відміну від MATLAB не можна одразу використовувати імена функцій без імпорту пакетів, що містять ці функції. Для імпорту пакетів необхідно скористатися оператором **import**, наприклад:

```
import numpy
```

Доступ до функцій пакету виконується через оператор «.»:

```
x = numpy.array([1, 2, 3, 4, 5])
```

Однак, такий запис може бути громіздким, тому імпорт пакетів доцільно виконувати, даючи їм скорочені назви-псевдоніми. Для цього використовується оператор **as**:

```
import numpy as np
```

Тоді доступ до функцій пакету numpy можна буде виконувати через скорочене ім'я «**np**», наприклад:

```
s = np.sum(x)
```

Ще приклади імпорту пакетів у Python:

```
import skimage.io as io
import scipy.signal as signal
import matplotlib.pyplot as plt
```

Введення матриць

Загалом мова Python має власні масиви та процедури роботи з ними. Однак, для більшої зручності спільнотою розробників було створено пакет, який називається numpy, що копіює функціонал середовища MATLAB. Справа в тому, що MATLAB зарекомендував себе з найкращої сторони при створенні алгоритмів та виконанні математичного моделювання. Його функціональні можливості при роботі з матричною алгеброю виявилися найбільш зручними і саме тому він використовується як взірець при створенні ряду нових мов програмування та їхніх бібліотек.

Введення матриць в Python може виконуватись за аналогією з MATLAB наступним чином:

```
A = np.array([[16,  3,  2, 13],
              [ 5, 10, 11,  8],
              [ 9,  6,  7, 12],
              [ 4, 15, 14,  1]])
```

Якщо необхідно одразу визначити тип даних матриці (за замовчуванням використовується **int32**), то це можна зробити після визначення елементів масиву, наприклад:

```
A = np.array([[16,  3,  2, 13],
              [ 5, 10, 11,  8],
              [ 9,  6,  7, 12],
              [ 4, 15, 14,  1]], 'double')
```

Звернення до елементів масивів

На відміну від MATLAB у мові Python індексація починається з починаючи з 0 та виконується спочатку не по стовпцям, а по рядкам, так само як у мовах C/C++. Для звернення до необхідного використовують квадратні дужки, в яких аналогічно до MATLAB вказується індекси необхідного елементу. Наприклад:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
[[1 2 3]
 [4 5 6]
 [7 8 9]]

A[1, 1]
5

A[0, 2]
3
```

Якщо необхідно отримати частину масиву (підмасив), то в дужках можна наводити масиви, що містять положення необхідних елементів вздовж кожної розмірності. Для автоматичного генерування цих масивів можна використовувати діапазони, користуючись оператором «:». Тут все

дуже подібно до MATLAB, але є і відмінності (зверніть увагу на останній приклад)

```
A = np.array([[17, 24,  1,  8, 15],
              [23,  5,  7, 14, 16],
              [ 4,  6, 13, 20, 22],
              [10, 12, 19, 21,  3],
              [11, 18, 25,  2,  9]])

B = A[1:4, 1:5]  # Останній елемент індексу не враховується
print(B)
[[ 5,  7, 14, 16],
 [ 6, 13, 20, 22],
 [12, 19, 21,  3]]

B[np.ix_([0, 2], [1, 3])]
array([[ 7, 16],
       [19,  3]])

B[[0, 2], [1, 3]]
array([7, 3])
```

Деякі корисні при роботі з матрицями функції

`np.tile(...)`	утворює матрицю, яка складається із повторень матриці-аргумента заданої кількості раз у заданих розмірностях (аналог функції **`repmat`** у MATLAB);
`np.reshape(...)`	змінює розміри матриці;
`np.meshgrid(...)`	генерує двомірну сітку;
`np.rot90(...)`	розвертає матрицю проти годинникової стрілки на 90 градусів;
`np.flipud(...)`	перевертає двомірну матрицю зверху вниз так, що перший рядок стає останнім, а останній – першим;
`np.fliplr(...)`	перевертає двомірну матрицю зліва на право так, що лівий стовпчик стає правим, а правий – лівим;
`np.moveaxis(...)`	змінює порядок розмірностей у багатовимірному масиві (аналог функції **`permute`** у MATLAB)
`np.roll(...)`	циклічний зсув елементів у матриці (аналог функції **`circshift`** у MATLAB);

MATLAB має ще величезну кількість функцій для зручної обробки матриць. З ними можна детально ознайомитись у допомозі системи MATLAB.

Для того щоб дізнатися яку розмірність та розміри має багатомірний масив чи матриця корисними можуть бути наступні функції:

`np.shape(...)`	повертає вектор кількостей (кількість) елементів вздовж кожної (заданої) розмірності; може змінювати розмір масиву (аналог функції **`size`** у MATLAB)
`np.size(...)`	загальна кількість елементів у масиві одразу по всім розмірностям (аналог функції **`numel`** у MATLAB)
`np.ndim(...)`	кількість розмірностей у масиві, що є аргументом даної функції

Наприклад:

```
a = np.ones([2, 3, 5])

np.ndim(a)
3

np.shape(a)
(2, 3, 5)

a.shape      # Те саме, що й попередній запис
(2, 3, 5)

np.size(a)
30

a.shape = (15, 2)
print(a)
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
```

Операції вводу-виводу зображень

Читання зображень в Python можна здійснити багатьма способами, але маючи бібліотеку scipy будемо користуватися саме нею. Виведення зображень виконуватимемо за допомогою пакету matplotlib. Щоб скористатися обома пакетами їх як завжди спершу треба імпортувати:

```
import skimage.io as io
import matplotlib.pyplot as plt
```

Основні процедури для роботи з введенням-виведенням зображень:

`io.imread('filename')`	завантаження зображення.
`io.imsave('filename', f)`	збереження зображення.
`plt.imshow(f)` `plt.show()`	відображення зображення.

Приклад:

```
img = io.imread('cameraman.tif');
io.imsave('myimage.png', img);
plt.imshow(img, cmap = plt.cm.gray);
plt.axis('off')
plt.show()
```

Після виконання функції **imshow** на екрані має з’явитись зображення показане на рис. Б.3.

Рисунок Б.3 – Виведене функцією **plt.imshow** зображення

ЛІТЕРАТУРА

1. *Гонсалес Р.*, Вудс Р. Цифровая обработка изображений. – М.: Техносфера, 2005. – 1072 с.
2. *Гонсалес Р.*, Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. М.: Техносфера, 2006. – 616 с.
3. *Абламейко С.В.*, Лагуновский Д.М. Обработка изображений: технология, методы, применение. Учебное пособие. – Мн.: Амалфея. – 2000. – 304 с.
4. *Форсайт Д. А.*, Понс Ж. Компьютерное зрение. Современный подход. / Пер. с англ. – М.: Издательский дом «Вильямс». – 2004. – 928 с.
5. Методы компьютерной обработки изображений / Под ред. В.А. Сойфера. – 2-е изд., испр. – М.: ФИЗМАТЛИТ. – 2003. – 784 с.
6. *Шапиро Л.*, Стокман Дж. Компьютерное зрение / Пер. с англ. – М.: БИНОМ. Лаборатория знаний. – 2006. – 752 с.
7. *Дьяконов В.П.* MATLAB 6/6.1/6.5 + Simulink 4.5. Основы применения. – М.: СОЛОН–Пресс. – 2004. – 768 с.
8. *Кухарев Г.А*. Биометрические системы: Методы и средства идентификации личности человека. СПб.: Политехника. – 2001. – 240 с.
9. *Прэтт У*. Цифровая обработка изображений / в 2-х. кн. – М.: Мир. – 1982. – 792 с.
10. *Хорн Б. К.П.* Зрение роботов. – М.: Мир. – 1989. – 400 с.
11. *Медведев В.С.*, Потемкин В.Г. Нейронные сети. MATLAB 6/ Под общ. ред. к.т.н. В.Г. Потемкина. – М.: ДИАЛОГ-МИФИ. – 2002. – 496 с.
12. *Комарцова Л.Г.*, Максимов А.В. Нейрокомпьютеры: Учеб. пособие для вузов. – М.: Изд–во МГТУ им. Н.Э. Баумана, 2002. – 320 с.
13. *Круглов В.В.*, Борисов В.В. Искусственные нейронные сети. Теория и практика. – 2–е изд., стереотип. – М.: Горячая линия-Телеком, 2002. – 382 с.

14. *Дьяконов В.П.* MATLAB 6.5 SP1/7/7 SP1 + Simulink 5/6. Работа с изображениями и видеопотоками. – М.: СОЛОН-Пресс, 2004. – 768 с.
15. Hoffmann G. CIE Color Space. Режим доступу: http://www.fho–emden.de/~hoffmann/ciexyz29082000.pdf. – перевірено 05.09.2011